Multi-criteria Optimal Task Allocation at the Edge

Kostas Kolomvatsos, Christos Anagnostopoulos

School of Computing Science, University of Glasgow
Glasgow, UK
e-mails: {Kostas.Kolomvatsos, Christos.Anagnostopoulos}@glasgow.ac.uk

Abstract

In Internet of Things (IoT), numerous nodes produce huge volumes of data that are subject of various processing tasks. Tasks execution on top of the collected data can be realized either at the edge of the network or at the Fog/Cloud. Their management at the network edge may limit the required time for concluding responses and return the final outcome/analytics to end-users or applications. IoT nodes, due to their limited computational and resource capabilities, can execute a limited number of tasks over the collected contextual data. A challenging decision is related to which tasks IoT nodes should execute locally. Each node should carefully select such tasks to maximize the performance based on the current contextual information, e.g., tasks' characteristics, nodes' load and energy capacity. In this paper, we propose an intelligent decision making scheme for selecting the tasks that will be locally executed. The remaining tasks will be transferred to peer nodes in the network or the Fog/Cloud. Our focus is to limit the time required for initiating the execution of each task by introducing a two-step decision process. The first step is to decide whether a task can be executed locally; if not, the second step involves the sophisticated selection of the most appropriate peer to allocate it. When, in the entire network, no node is capable of executing the task, it is, then, sent to the Fog/Cloud facing the maximum latency. We comprehensively evaluate the proposed scheme demonstrating its applicability and optimality at the network edge.

Keywords: Edge-centric computing, task allocation, multi-criteria decision making

1. Introduction

The Internet of Things (IoT) gives the opportunity for the creation of intelligent applications on top of numerous computing, sensing, and actuation devices. Devices are interconnected to communicate while they collect data from their environment and process them becoming knowledge producers. Knowledge may have the form of a response in analytics queries (predictive and inferential analytics) defined by end-users or applications [35], or they may be the result of more complicated tasks. Legacy systems adopt the Cloud infrastructure where various services' models are available. However, large-scale data centers, present in Cloud, are centralized systems which implies a large average separation between devices and Cloud. This, in turn, increases the average network latency and jitter negatively affecting delay sensitive, real-time applications [40].

For alleviating the problem of delay, Edge Computing (EC) [38] is coming into scene. In EC, numerous human-controlled devices form the network edge like tablets, smart-phones, sensors or nano data-centers [16]. Edge-centric computing aims at a distributed model that interconnects heterogeneous resources controlled by various nodes to the Fog and, accordingly, to the Cloud. In EC, we can deploy Cloud-like capabilities in the devices to make them able to process the collected data. Hence, we minimize the required time for initiating tasks processing and acquiring responses. Fog Computing (FC) is the next step, one hop away for the data production and the aforementioned pre-processing model. In FC, nodes communicate with cloudlets, and cloudlets communicate with Cloud to realize the data processing and analytics tasks. Both EC and FC aim to keep processing tasks close to the nodes, thus, the sources of contextual data to limit the latency in the provision of responses. Recent advances in the field involve the adoption of light weight virtualization techniques on top of the available heterogeneous devices present at the EC [29]. The orchestration of the devices could be realized through containers or unikernels facilitating the packaging of the provided services [33]. A comprehensive performance evaluation that aims to show the strengths and weaknesses of several low-power devices when handling container-virtualized instances is presented in [28] while their suitability is reviewed in [34]. Other recent approaches incorporate P2P control protocols for the exchange of service provisioning information between various FC nodes and their coordination [39].

For supporting IoT applications, edge nodes should perform/execute a

set of tasks. Tasks are generated, possibly, at high rates and should be concluded immediately in terms of allocation and execution. In the relevant literature (see next Section), task allocation and scheduling originates in the management of a group of nodes. The allocation is, then, adopted to determine the assignment of each task to a node while scheduling mainly aims to the sequence of the execution for each task. The challenges are to **(C1)** maximize the performance and **(C2)** minimize the energy consumption, thus, maximizing the lifetime of the network. Multiple research efforts deal with centralized approaches, thus, the allocation and scheduling models suffer from the drawbacks reported in the literature for Cloud computing. In this paper, we build on the *autonomous* nature of nodes and propose a distributed scheme for *pushing* the local optimum allocation of incoming tasks from centralized decision making to the network edge. In contrast to other research efforts elaborated in Section 2, we consider that each task can be **(i)** executed in an edge node itself, or **(ii)** executed in a group of peer/neighboring nodes, or **(iii)** delegated to the Fog/Cloud. Due to energy and computational constraints, each node can afford a limited number of tasks; it should select those that *maximize* its performance while meeting certain resource constraints. The rationale behind our scheme is that we distinguish two conditional decisions on an edge node: the first decision is related to whether a task ***can be executed locally***. The second decision is related to whether the task ***can be executed in the group of peers*** or in the Fog/Cloud conditioned on the result of the former decision. The aim is to *optimally* decide the execution of tasks starting from the node itself to minimize the time for initiating the execution. The first decision is achieved by a classification model derived from an k-Nearest Neighbor Classifier (kNNC) [20]. The conditional second decision is obtained by adopting the utility theory [17]. For both decisions, we aim to find the closeness of the incoming tasks with (in a sequential order): (i) the training set adopted to indicate which tasks should be executed locally based on a set of constraints; (ii) the characteristics of the peers in the network. Both decisions are made over contextual information related to the status/context of the node, e.g., current load, remaining resources, collected data distribution, and every task's characteristics, e.g., priority, execution requirements.

The paper is organized as follows: Section 2 reports on the related work and how our mechanism departs from it discussing the key contribution points. Section 3 presents the problem of pushing the task allocation to the edge and the decision making methodology. In Section 4, we describe

the proposed in-network centric approach while in Section 5, we provide an analysis for the short- and long-term load of nodes. In Section 6, we present the experimental evaluation of the proposed scheme and Section 7 concludes the paper discussing our future research plans.

2. Related Work & Contribution

Tasks allocation and scheduling are important research subjects in multiple domains. Both subjects have a significant impact in the IoT and EC as nodes have limited computational capabilities while being restricted by various energy constraints. Hence, nodes should carefully select the tasks that they will execute locally making imperative the need for applying intelligent techniques for tasks scheduling. In any case, nodes should take into consideration tasks' specific characteristics in combination with their current status for any decision making. Cloud serves as the intermediate between IoT devices and applications exhibiting a vast infrastructure where increased computational (possibly virtualised) capabilities are available for processing. EC provides the necessary framework for hosting and executing tasks with the minimum delay/latency. Smart gateways and micro-data centers are adopted to facilitate the task processing [1], [19]. A widely studied research subject is task scheduling in Wireless Sensor Networks (WSNs). Task mapping and scheduling should take into consideration energy constraints to secure an efficient execution [48], [6]. A task pre-processor and a scheduler can be responsible for the final allocation. The pre-processor tries to identify the energy requirements of the incoming tasks and, based on energy monitoring activities, decides on the final scheduling. Another approach is to study a fair energy balance among sensors while minimizing the delay using a market-based architecture [12]. Nodes are modeled as sellers communicating a deployment price for a task to the consumer. Taking into consideration the defined constraints, nodes may cooperate to conclude the final allocation of tasks [2]. Example algorithms involve task clustering and node assignment mechanisms based on task duplication and migration schemes. In any case, the aim is to minimize the execution time, thus, to deliver the final response in limited time [10]. A model that could be adopted for such purposes is to cluster the network and build intra-cluster and inter-cluster scheduling relations. An Integer Linear Programming (ILP) formulation and a 3–phase heuristic are also adopted to solve the task allocation problem in [57]. Each sensor node is equipped with discrete Dynamic Voltage Scaling (DVS) while

the time and energy costs of both computation and communication activities are considered. In [56], the authors propose a modified version of binary Particle Swarm Optimization (PSO). The method adopts a different transfer function, a new position updating procedure and mutation for the task allocation problem. Another PSO-based solution is presented in [37] which allocates tasks into a number of robots trying to decrease the communication cost. In [22], the authors present a task allocation mechanism of a dynamic alliance that is based on a Genetic Algorithm to acquire the balance between energy consumption and accuracy. In [9], the authors discuss three algorithms to solve the task allocation problem: a centralized, an auction-based, and a distributed algorithm. The distributed algorithm adopts a spanning tree over the static sensors to assign tasks.

A set of sub-tasks may consist of a more generic task. The efficient combination and execution of sub-tasks will lead to the efficient completion of the initial, more generic, task. A scheduling algorithm for a set of sub-tasks is proposed in [51]. The aim is to assign each sub-task into the available nodes for maximizing the performance. Zenith [54] proposes a methodology for resource allocation in a set of small-scale micro-data centers that represent an ad-hoc and distributed collection of a computing infrastructure. Zenith allows service providers to establish resource sharing contracts with the edge infrastructure.

Task scheduling heavily depends on the application domain. IoT and Cloud define different requirements due to their different nature. Cloud mainly offers the available services on demand, while the IoT may involve applications where nodes push their data and knowledge into the network. A review on scheduling algorithms that fit on both the Cloud and the IoT is presented in [41]. In [35], the authors propose a model for data distribution over IoT devices. The model aims to eliminate bandwidth and storage constraints. Simulated annealing is also adopted to solve the problem of scheduling while optimization techniques can be used on top of the load of tasks in Cloud applications [30]. The model tries to build on the parallelization of allocating various tasks in a multi-cloud system. Another task allocation scheme for Cloud is presented in [58]. The provided model takes into consideration both task and nodes diversity. Workloads are clustered into different classes with the same characteristics through the adoption of the k-means algorithm. In [25], the authors propose a Quality of Service (QoS) aware resource scheduling algorithm adopting a PSO model to derive the final scheduling. The aim is to reduce the required time for deciding

the final allocation and ensure the load balancing towards the maximization of the performance. Finally, JarvSis is proposed as a distributed scheduler capable of automating the execution of multiple heterogeneous tasks in IoT [5]. Through JarvSis, developers can easily configure and deploy hierarchies of control tasks.

Recently, research community focuses on the management of virtualized resources and their combination to perform processing at the edge of the network. Some efforts incorporate algorithms for the allocation of the processing tasks to the available nodes. For instance, in [46], the authors propose a novel workflow-like service request scheme and a dynamic algorithm for allocating the virtualized resources to multiple processing points. The aim is to map the requests defined in a workflow format to the services offered by the edge computing. The provided simulations reveal the minimum latency and an efficient behavior when uploading the involved virtualized resources. In addition, the research effort presented in [47] aims at proposing a model for the management of Cloud-of-Things and Edge Computing (CoTEC) traffic in multi-domain networks. In this effort, the proposed scheme incorporates traditional multi-topology routing and introduces a number of programmable nodes that can be configured to ease the ongoing traffic. Routing tasks are allocated to the available nodes while the provided results show a lower execution time and a better quality of service compared to a model that does not adopt the proposed algorithm.

Studying other relevant efforts in the field, we observe that the majority of them focus on the WSNs domain. Hence, the main attention is paid on energy constraints when finalizing the allocation of tasks. In addition, they also focus on eliminating the communication overhead towards the reduction of messaging to lower the transmission collisions. They are, usually, *centralized* approaches, i.e., a central entity decides on the final allocation based on the currently available contextual information. This inevitably conveys the disadvantages of any centralized system, i.e., increased communication overhead and a single point failure. The central entity can also decide to transfer the data to the node where each task will be executed (with the use of migration algorithms). In general, migration techniques focus on 'univariate' contextual information. This means that the final decision is delivered taking into consideration only a single parameter/perspective, e.g., energy, transmission requirements, topology of the network. More importantly, data migration techniques suffer from the increased migration cost especially at the network edge due to the increased amount of data 'circulated' in the net-

work. To the best of our knowledge, our scheme is one of the first attempts that departs from the centralized task allocation intelligence and focuses on a distributed, local 'multivariate' scenario where the intelligence is pushed to the edge of the network. The final decision is locally made taking into account multiple parameters, e.g., the current load of nodes, their speed of processing, the communication cost, and the remaining resources. That is, our scheme casts as a local multi-criteria optimization mechanism for delivering the best decision. Apart from that, we further take into consideration the data collected at each node without adopting any data migration solutions, thus, avoiding redundant communication overhead. The proposed sequential decision making scheme aims to eliminate the initiation time of a task giving priority to the local execution, i.e., to the node where each task is initially reported.

Our model can be also combined with other schemes recently proposed for the management of tasks at the edge of the network. For instance, the virtualized resources allocation for processing [46] or the deployment of network services into a set of programmable router nodes [47] could be the first step before the execution of our scheme. Such efforts (i.e., [46], [47]) focus on an allocation based on a 'global' view on the available processing points / nodes deciding over the information available for the present nodes. This information is related to the network performance. Our scheme could consist of the second step where every node receiving a task could decide if it has the resources and the data to process the task locally. Hence, in this second step, we have the nodes deciding based on the 'local' view on their status and their peers. Actually, the output of the algorithms like those provided in [46] & [47] could be the starting point for our model triggering the efficient management of the incoming tasks.

The following list summarizes the contributions of our paper:

- we provide a distributed, 'multivariate' decision making mechanism for the optimal allocation of tasks;
- our model 'reasons' over the status and the data present in each autonomous node;
- our mechanism does not require the migration of data to become the subject of tasks' execution, thus, we avoid the redundant communication overhead i n the network;

- the proposed mechanism decides based on the 'local' view of the status of each node;
- we provide a large set of simulations adopting real and synthetic data together with a comparative assessment with other models in the domain.

3. High Level Description of the Proposed Scheme

We consider a set of IoT nodes, i.e., $\mathcal{N} = \left\{n_1, n_2, \ldots, n_{|\mathcal{N}|}\right\}$ responsible to 'observe' their environment and collect contextual data. On top of the collected data, nodes can execute a set of (simple) processing tasks. A task stream $\mathcal{T}_i$ reported to node n_i is defined by a series of ordered tuples $\langle t, T_{it}, \mathcal{C}_{it}\rangle$, where t is the time-stamp of the task T_{it} and $\mathcal{C}_{it}$ is the set of constraints for T_{it}. The processing of each task corresponds to a result that is delivered to end users or applications. We consider that every task is accompanied by a set of constraints $\mathcal{C} = \left\{c_1, c_2, \ldots, c_{|\mathcal{C}|}\right\}$. For instance, let $\mathcal{C} = \{latency, lifetime\}$ be the set of constraints and $\mathcal{C}_{it} = \{2.0, 15\}$ is their realization reported for T_{it} at some time instance t. When no constraints are present, then $\mathcal{C}_{it} = \emptyset$. Among the characteristics/constraints of a task, in this paper, we focus on its priority and complexity. Such parameters depict two significant aspects of a task execution process, i.e., an indication of the immediate initiation of its execution (priority) and the time and resources required for the execution (complexity).

Constraints can be in any form, however, a methodology that matches them to nodes' characteristics is necessary. We could consider constraints related to libraries that should be adopted by nodes when executing a task, we could involve intervals or non-linear relations and match them with the specific characteristics of each node. Constraints can be incorporated in an 'aggregation' function that will result a subset of them that will be taken into consideration in the final processing. An aggregation function could incorporate the strategy that we want to adopt when deciding to execute each incoming task. For instance, the involvement of specific libraries or non-linear relations between constraints may increase the execution complexity of a task with specific consequences in the decision making. The aforementioned approach is part of our future research plans. In Figure 1, we show an example environment with $|\mathcal{N}| = 4$ nodes. After the reception of task through $\mathcal{T}_i$, n_i should decide if it should execute it locally or transfer it to its

peers/Fog/Cloud. Specifically, n_i should sequentially decide on the following actions: **Action 1.** Execute T_{it} locally; **Action 2.** Send T_{it} for execution to one of the peers present in the same local network; **Action 3.** Send T_{it} to be executed in the Fog/Cloud. Actually, these actions could be seen as the result of two sequential decisions, i.e, **D1**. Decide if n_i can execute T_{it}; **D2**. If not, decide if there is a peer node to 'host' T_{it}. Actions 2 and 3 are examined conditioned to the decision for Action 1.

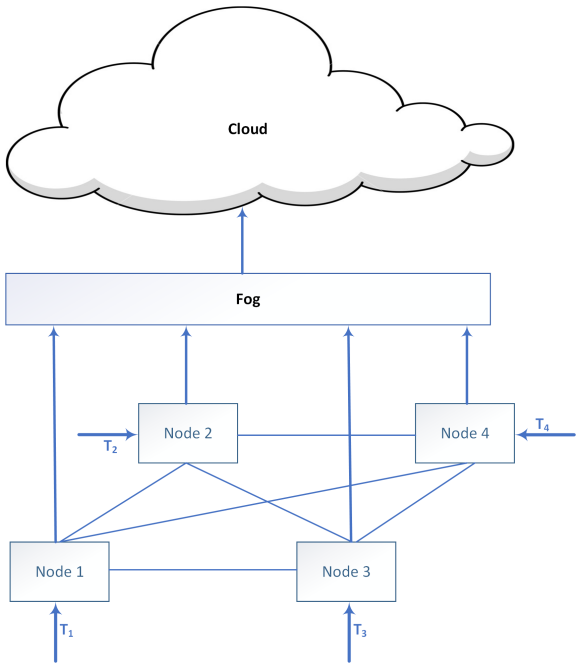


Figure 1: An example of an edge-network architecture with task flows among edge nodes.

Locally, n_i after the reception of T_{it}, it concludes, based on its current status and T_{it}'s requirements a *Task Requirements Tuple* (TRT) i.e., $\langle l_t, r_t, z_t, b_t \rangle$, where l_t is the current load, r_t is the remaining resources, z_t is the priority of the task and b_t is the complexity of the task; $l_t, r_t, z_t, b_t \in [0, 1]$. In this paper, we rely on the priority and complexity of a task a representative characteristics/constraints for indicating the time and resources requirements that should be met when allocating tasks to the available nodes. Without loss of generality, we consider that l_t and r_t represent, with real values, the current load and the resources left for tasks execution, respectively. Both can be delivered by specific processes (their presentation is beyond the scope of the current paper). z_t can be also depicted in the interval $[0, 1]$ by dividing it into

equal sub-intervals. For instance, if we want to incorporate four priorities, the first could be depicted by the value 0.25, the second by the value 0.50 and so on. In addition, b_t represents the complexity of each task related to the required calculations to conclude the final result. b_t is 'profiled' in each task and its calculation is beyond the scope of the current work. However, we could also separate the interval $[0, 1]$ into equal sub-intervals as in the z_t case. For instance, if we focus on the following complexities, $nlogn, n^2, 2^n$, the first could be depicted by 0.33, the second by 0.66 and the third by 1.00. In this approach, the available complexities should be sorted in an 'increasing' order.

The discussed nodes form a graph $G = (\mathcal{N}, E)$ where E is the set of edges connecting the nodes. Each connecting edge $e_{ij} \in E$ defines the communication channel between nodes n_i and n_j and is characterized by a communication cost $\kappa_{ij} \in \mathbb{R}^+$. At pre-defined intervals, nodes exchange information about their status including their load and remaining resources to support the distributed decision making process. The discussed message is in the form $\langle l_j, r_j, \tau_j \rangle$ where the index j refers to the jth node. τ_j is defined through the calculation of the number of tasks successfully concluded in a time interval (i.e., the throughput of the node). The message is processed locally to create a tuple for each peer. Initially, n_i checks if T_{it} can be executed locally, thus, the communication cost is $\kappa_{ii} = 0$ and the time for starting the execution is limited (decision D1 - Action 1). The first decision is made using the TRT which represents the context of n_i and the requirements of the task. If the decision is negative, n_i checks if T_{it} can be executed by its neighborhood $\mathcal{N} \setminus \{n_i\}$. The second decision is based on the tuples $\langle l_j, r_j, \tau_j, \kappa_{ij} \rangle$ as derived by the reported messages. In such case, the cost for starting the execution is analogous to the κ_{ij}. It should be noted that κ_{ij} is dynamically adapted to the network conditions. When nodes exchange their statuses at pre-defined intervals, they also conclude the cost κ_{ij} with the 'assistance' of the aforementioned messages maintaining the calculated historical values for future use. The second decision is based on the distance between the task characteristics and the status of each node accompanied by the communication cost. Again, if no node could be selected for assigning T_{it}, n_i decides to send T_{it} to Fog/Cloud with an increased communication cost; higher than the $\max(\kappa_{ij}), \forall j$. In Figure 2, we present the discussed internal decision process.

Motivating Example. Assume that we want to monitor a forest for detecting emergency situations like fires. A set of nodes are placed in the forest and are responsible to collect various data like temperature, humidity,

etc. Nodes are characterized by specific resources and can stored limited amounts of data (usually, a window of the collected measurements) for performing a simple processing. The area covered by the forest, is separated into a number of sub-areas, thus, a set of nodes may observe the same sub-area. In the back end system placed at the Cloud, there is the opportunity for end users to define their queries and get information about the current status of the forest. These queries may be separated into a number of sub-queries, i.e., tasks that should be responded by the available nodes. When a query is fired, its sub-queries are reported to a node that should respond as soon as possible. For instance, the node can be instructed to report the average measurements for a time interval or to apply regression models and so on. Every node after the reception of each task checks its resources, its load as well as task's characteristics and decide if it will be executed locally. If not, the node selects the most appropriate peer (a node located in the same sub-area) to allocate the task for execution. Otherwise, the node sends the task to an application performing mathematical calculations placed in the Cloud and wait for the final response facing increased latency that will affect the final response time.

As mentioned, at pre-defined intervals, nodes exchange their load, remaining resources and speed to provide a view on their status to peer nodes. These intervals should not be low as the network will be flooded by the performance messages affecting the communication cost κ, however, they should not be high as nodes will not have an 'fresh' view on the performance of their peers. In any case, in the time between the intervals possible changes may happen in the performance of nodes; it consists of a stochastic process, thus, nodes cannot have a view on the completion time of each task that may vary. Furthermore, for eliminating the completion time we can increase/enhance nodes' characteristics/resources, however, this is very difficult to happen in a 'working' network. Let x is the time observed to get a response for a task. We also get z as the waiting time for a task to be executed and q as the completion time. The following equation stands true: $x = z + q$. Both z and q are stochastic variables affected by various parameters. For instance, z can be affected by the number of tasks waiting in the execution queue. Suppose, we are able to estimate q either a task is executed locally or in a peer or in Fog/Cloud. Then, x heavily depends on z. There are two approaches that may be adopted in our scenario. The decision for the local execution is made on top of **(i)** q or **(ii)** z. When the decision is based on q, we try to minimize the time for starting and executing future tasks, i.e., this is a

type of a priority scheme. In this scheme, the task with the lowest q has the highest priority and it will be executed first. Such an approach is typical in operating systems for the management of processes [42]. However, it is known that priority schemes suffer from starvation, i.e., indefinite blocking of tasks that exhibit high completion time. When the decision for a task execution is made based on z, we have the following choices: (i) execute it locally with z being the time for which the task will wait in the local execution queue; (ii) execute it in a peer with z being equal to the transmission time plus the time for which the task will wait in the peer's execution queue; (iii) execute it in the Fog/Cloud with z being equal to the transmission time plus the latency for starting the execution of the task and getting the final result. However, Cloud offloading almost always incurs an additional 100 to 200 ms latency compared to using edge computing solutions [7]. In this paper, our view is that tasks should be processed in a sequential order to avoid possible starvation effects and propose a model that tries to eliminate the waiting time before the execution of a task starts. An hybrid solution, i.e., a model that focuses not only on the elimination of z but, in parallel, in handling a priority scheme (based on the lowest q) is part of our future research agenda.

The proposed decision making mechanism is a function

$g : \{TRT_t, \langle l_j, r_j, \tau_j, \kappa_{ij} \rangle\} \rightarrow \{A\}$ where A is the set of the available actions defined as follows: $A = \{a_1, a_2, a_3, \ldots\}$. In our case, $A = \{n_i, \{n_{selected}, \emptyset\}\}$. n_i is the node deciding for T_{it} and $n_{selected}$ is the peer selected to host/execute T_{it} when it is subject of a transfer. A formal definition of the available decisions is:

- **Decision 1.** Apply the current TRT in the decision making mechanism g and decide if T_{it} can be locally executed; possible actions $\{a_1 = n_i, \{a_2 = n_{selected}\}\}$.

- **Decision 2.** If T_{it} cannot be locally executed, decide the peer node where T_{it} will be transferred for execution; possible actions $\{a_2 = n_{selected}, a_3 = \emptyset\}$. If no peer is appropriate for executing T_{it}, send T_{it} to the upper layer (Fog/Cloud); action $\{a_3 = \emptyset\}$.

Proposition 1. Function g concludes the one-step optimal execution of T_{it} w.r.t. actions a_1, a_2, a_3 based on TRT_t and context vectors $\langle l_j, r_j, \tau_j, \kappa_{ij} \rangle$.

Proof: g delivers the final action a_i based on a sequential processing. It applies the *principle of optimality* [36], which implies that every decision

should be optimal for the remaining problem (initially, we select between three actions, next, we select between two actions). Actually, g applies the one-step optimality process. As the time and remaining resources are the critical parameters, g, initially, examines the possibility of executing T_{it} locally taking into consideration the parameters l_t, r_t, z_t, b_t and communication cost $\kappa_{ii} = 0$. The local execution of T_{it} secures that the time required for starting T_{it} is limited. In addition, the decision is made taking into consideration the load and remaining resources. Hence, if the node has enough resources to conclude on T_{it} will decide the local execution. On top of these parameters, it derives the optimal result which is one of: $\{n_i, \mathcal{N} \setminus \{n_i\}\}$. When, the decision $\{\mathcal{N} \setminus \{n_i\}\}$ is made, g examines the execution of T_{it} in the neighboring nodes to (again) limit the starting time. In this case, there will be a communication/transfer cost in terms of time and resources. However, g derives the result which will be one of the following: $\{n_{selected} \in \mathcal{N} \setminus \{n_i\}, \emptyset\}$. The decision is optimal in terms of time and resources as the execution of T_{it} in $n_{selected} \in \mathcal{N} \setminus \{n_i\}$ is concluded based on the T_{it} and $n_{selected}$ characteristics. g applies a utility maximization decision making for optimally deciding the appropriate action. When $\{\emptyset\}$ is the final decision, T_{it} will be transferred to the Fog/Cloud which is, again, the optimal decision as no node in the group can efficiently execute T_{it} based on the set of realized constraints and task's/nodes' characteristics. □

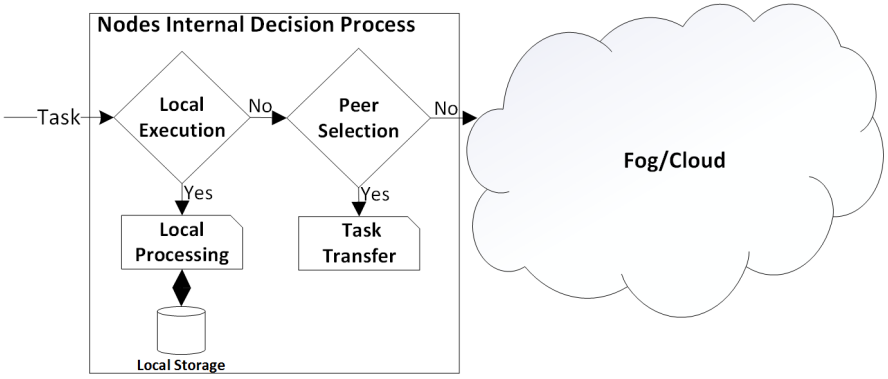


Figure 2: The internal decision process of each node. A sequential decision making is fired for each incoming task.

4. The Task Allocation Scheme

Every node should apply the proposed scheme in a number of tasks arriving through streams. For the realization of the proposed scheme, we rely on techniques that take into consideration the combination of the available contextual data (the aforementioned tuples) before finalizing the outcome. The adopted techniques can deliver the result in the minimum time as we aim to support (near) real-time IoT applications. For concluding on the **Decision 1**, we rely on the k-Nearest Neighbors (kNN) classification [20], while for the **Decision 2**, we adopt the principles of utility theory [17]. Our previous work presented in [24] deals with the same problem as our current effort, however, it solves it through the adoption of a capacity model. In [24], we propose a scheme for selecting the most significant tasks to be executed at the edge providing a model for defining the significance level of a task and taking into consideration the energy constraints of nodes. Usually, capacity schemes suffer from the conversion process into the mathematical model. In addition, it is difficult to 'aggregate' multiple parameters into the same model and get the final result in a reasonable time (it depends on the number of parameters). Our current effort 'sees' the problem as a process that classifies a task into two classes, i.e., the local execution or the transfer to another node/Fog/Cloud. In the respective literature, one can find a number of research efforts that handle a resource allocation model as a classification problem [8], [15], [31], [45], [52]. The advantage is that classification models may incorporate the relationships between the adopted parameters and can be based on past experiences as represented by historical values. In addition, the separation of the solution environment into a set of classes, it can simplify the environment reducing the confusion about the appropriate solution.

4.1. kNN Classification

The local decision making depends on a training dataset and a kNN Classifier (kNNC). We select such a technique, as kNNCs exhibit ease interpretation, low calculation time and acceptable predictive power when compared with other techniques, e.g., logistic regression, random forests. In addition, a kNNC is non parametric, which means that it makes no explicit assumptions about the form of the function producing the final result. However, it heavily depends on the provided training dataset. The kNNC decides whether the node receiving T_{it} can/should execute it locally or not. This decision is made based on the TRT_t i.e., $\langle l_t, r_t, z_t, b_t \rangle$.

The kNNC is based on learning by analogy, i.e., it compares the given tuple with the training tuples to identify the similar ones. The training tuples and the incoming tuples are characterized by the same number of variables/attributes. The training tuples can be extracted by a statistical analysis process performed on top of historical values. Such historical values are recorded (e.g., for a warm up period) and adopted to build the classifier. It becomes obvious that, in this process, the intervention of experts that will define the final class for each tuple is required. The kNNC searches in the dimensional space (let Y be the number of variables/dimensions), the pattern space that is close to the incoming tuple. We consider that closeness is defined through the adoption of the Euclidean distance, i.e., $d(\mathbf{TRT}_t, \mathbf{TRT}_s)$, $\forall s$; s is the index of each training tuple met in the available dataset D. Actually, D consists of multiple TRTs accompanied by the appropriate action for each one (local execution or not). Other distance measures could be also adopted e.g., Manhattan, Minkowski, Hamming. Let y_i be the ith variable in the TRTs (i.e., $y_i \in \{l, r, z, b\}$). Then, the aforementioned Euclidean distance is defined as: $d(\mathbf{TRT}_t, \mathbf{TRT}_s) = \sqrt{\sum_{i=1}^{|D|} \left(y_{i,TRT_t} - y_{i,TRT_s}\right)^2}$. The closeness between the incoming tuple and the training tuples is based on the difference of the numeric values for each variable. The kNNC estimates the conditional probability for each class. There are two classes in our case; the local execution depicted by the action $a_1 = \{n_i\}$ and the transfer to peers depicted by the action $a_2 = \{\mathcal{N} \setminus \{n_i\}$. The probability is translated as the fraction of points with the corresponding class label. The incoming tuple is assigned to the most common class among the k nearest neighbors i.e., the class with the highest probability. The value for k is selected to be the value that minimizes the error rate and it is derived through simulations.

4.2. Peer Selection Scheme

The second decision of our scheme is related with the identification and the selection of the appropriate node for allocating any 'rejected' task. For this decision, we rely on the principles of the Utility Theory [17]. We focus on the multi-attribute utility theory [23] where we consider that each peer is characterized by the above discussed tuples $\langle l_j, r_j, \tau_j, \kappa_{ij} \rangle$. In this section, the notion of 'attribute' is the same with the notion of variable/dimension adopted in the previous sections. As every node is characterized by multiple attributes, we aim to 'combine' them and provide a final ranking to select the most appropriate peer. Multi-attribute utility theory concerns with expressing the utilities of multiple attributes (called as individual utilities) as a

function of the utilities of each attribute taken singly [17]. Our approach focuses on the selection of the most desirable alternatives among many different alternatives. In theory, many functions can be adopted for the calculation of individual utilities. For instance, we could rely on additive, multiplicative or multi-linear functions. However, as studied in [23], for four or more attributes, the reasonable models are the additive and the multiplicative schemes. In our case, we adopt both of them and provide two rankings of the available peers. Afterwards, we aggregate the two ranked lists and select the node present in the first place of the final aggregated list. If no node in the final list exhibits an aggregated result above a pre-defined threshold, the task will be allocated for execution in the Fog/Cloud. The above described process is fired for every T_{it} for which the action a_2 is decided.

As mentioned, in n_i, there is available information for the status of peers, i.e., $\langle l, r, \tau, \kappa \rangle$. Our model can be easily extended to involve more attributes. For some attributes, we desire to enjoy a low value close to 0 to gain high utility (e.g., l, κ) while for others, we aim at high values close to unity to gain high utility (e.g., r, τ). The former attributes are called *non-proportional*, while the latter are called *proportional*. For each attribute y_s, $s = 1, 2, \ldots, Y$, we define the individual utility function $f(y_s)$ based on the exponential distribution. For proportional attributes, we get $f(y_s) = e^{-\gamma y_s + \delta}$ while for non-proportional attributes we get $f(y_s) = \frac{1}{e^{-\gamma y_s + \delta}}$. Parameters γ and δ are adopted to produce values in the interval $[0, 1]$ and affect the final utility. For instance, we can follow more 'strict' strategies where the utility abruptly falls to zero when an attribute exceeds a threshold or be more relaxed concerning that the tendency is smoother than in the previous example.

These single/individual utility functions are initially 'combined' with the additive function: $U_{ADD}\left(y_1, y_2, \ldots, y_Y\right) = \sum_{s=1}^{Y} w_s f(y_s)$, with $\sum_{s=1}^{Y} w_s = 1.0$. In the case of the multiplicative function, the following equation stands true: $U_{MUL}\left(y_1, y_2, \ldots, y_Y\right) = \prod_{s=1}^{Y} w_s f(y_s)$. In general, the definition of the appropriate weights is a strategic decision based on the attributes we want to pay increased attention when we calculate the weighted result. In the relevant literature, one can find various efforts that propose automated ways to calculate the most appropriate weights. Some of these models are the Min-Max principle [18], optimization approaches [43] or geometric programming [21], [44]. The adoption of an intelligent technique can facilitate the dynamic definition of weights according to the characteristics of the environment. However, this dynamic approach is beyond the scope of the curent work. Based on U_{ADD} and U_{MUL}, we provide two ranked lists of the peers

in a descending order. Peers in the first place of the lists offer the highest possible utility when T_{it} will be allocated to them.

The final step is the aggregation of the two lists, i.e., $\mathbf{U}_{ADD}$ and $\mathbf{U}_{MUL}$. For the provision of the final list $\mathbf{U}$, we rely on a simple and fast technique, i.e., the Borda count model [13]. We consider that the aforementioned lists are preferences of peers retrieved by the corresponding functions. The peer present in the first place of a list gets $|\mathcal{N}| - 1$ points, the second gets $|\mathcal{N}| - 2$ points and so on. The final list $\mathbf{U}$ is created through the averaging of the points collected by the two lists. $\mathbf{U}$ is, finally, sorted in descending order. If no node exhibits a result over a pre-defined threshold, the task is allocated to the Fog/Cloud otherwise, the first node will host T_{it}. The pre-defined threshold could be delivered through simulations or through a dynamic threshold optimization model to derive the most appropriate value for each instance of the problem [11], [14], [50].

5. Estimating the Load of Nodes

5.1. The Short Term Expected Load

As nodes process a number of tasks, their load will be affected by their report rate. The more the number of tasks, the higher the load becomes. Recall that the load of nodes affect the decisions related to the location of the execution of each task. In this section, we provide an analysis on the short term expected load for each node to have a view on the parameters that affect their performance. The short term expected load is related to the number of tasks that will be executed locally after their initial allocation. We consider that multiple 'execution eras' deal with the execution of tasks reported in G. At every era, nodes receive a number of tasks and apply the proposed sequential decision making process. Let us define the following events: **(i)** M1: T_{it} arrives at n_i; **(ii)** M2: n_i decides the action a_1. Without loss of generality, we consider that these events are independent. The decision for action a_1 (i.e., event M2 - the local execution of T_{it}) depends on the expected number of the incoming tasks at n_i and the probability of local execution. The arrival of tasks in G can be modeled as a Poisson process with rate λ while the decision for local execution can be seen as a Bernoulli trial with probability of success $p = P(M2)$. A Poisson process arises when there is a large number of sources that produce events independently, e.g., arrivals, requests to a server, etc [32]. In addition, a trial (e.g., a local decision making)

where only two outcomes are possible (e.g., local execution of a task or not) can be modeled as a Bernoulli trial [49].

Proposition 2. The expected number of tasks arriving at n_i is $\frac{\lambda}{|\mathcal{N}|}$.

Proof: We consider that $|T|$ tasks arrive in G with a rate λ in a time unit (e.g., an hour, a day, a week). The arrival of tasks in G follows a Poisson distribution while the initial 'allocation' of tasks follows a Uniform distribution, i.e., reporting every T_{it} to n_i. Based on the Poisson distribution, the expected number of tasks reported to n_i is: $\lambda\frac{1}{|\mathcal{N}|} = \frac{\lambda}{|\mathcal{N}|}$. □

n_i after the reception of T_{it} checks the available training dataset and applies the kNNC. The probability of the local execution, then, depends on the 'class' indicated by the majority of the k neighbors. Any decision for local execution indicates that the majority of the kNNs report the action a_1, thus, the 'class' n_i. Initially, we assume that values for each variable/dimension y follows the Gaussian distribution. In general, a continuous-valued variable is typically assumed to have a Gaussian distribution to easily estimate the distribution's parameters (i.e., mean and standard deviation) from training samples [20].

Lemma 1. The probability of locally executing T_{it} when the Gaussian distribution is assumed is

$p = \sum_{m=\lfloor \frac{k}{2}+1 \rfloor}^{k} \binom{k}{m} \left(\frac{\pi_1}{|\mathcal{N}|\pi_2}\right)^m \left(1 - \frac{\pi_1}{|\mathcal{N}|\pi_2}\right)^{k-m}$

where $\pi_1 = \prod_{\forall s} \frac{1}{\sigma_{n_i}} e^{-\frac{(y_s-\mu_{n_i})^2}{2\sigma_{n_i}^2}}$ and $\pi_2 = \prod_{\forall s} \frac{1}{\sigma_s} e^{-\frac{(y_s-\mu_s)^2}{2\sigma_s^2}}$.

Proof: For calculating p, we should calculate the probability of having the majority of the k neighbors indicating the class n_i. The probability of the majority among the k neighbors is [26]: $P(majority) = \sum_{m=\lfloor \frac{k}{2}+1 \rfloor}^{k} \binom{k}{m} (p')^m (1-p')^{k-m}$ where p' is the probability of having the mth neighbor at the correct class (i.e., the tuple indicating local execution - class n_i. Under the assumption that variables/dimensions follow the Gaussian distribution, the probability of having the TRT_t 'generated' by the n_i is $P(n_i|TRT_t)$. Based on the Bayes theorem, we get: $P(n_i|TRT_t) = \frac{P(TRT_t|n_i)P(n_i)}{P(TRT_t)}$. However, $P(n_i) = \frac{1}{|\mathcal{N}|}$. In addition, $P(TRT_t) = \prod_{\forall s} \left(\frac{1}{\sqrt{2\pi}\sigma_s} e^{-\frac{(y_s-\mu_s)^2}{2\sigma_s^2}}\right)$ where μ_s and σ_s are the mean and the deviation of the sth variable/dimension as calculated through the training dataset. In addition, y_s is the sth variable in the TRTs (i.e., $y_s \in \{l, r, z, b\}$). Finally, we get $P(TRT_t|n_i) = \prod_{\forall s} \left(\frac{1}{\sqrt{2\pi}\sigma_{n_i}} e^{-\frac{(y_s-\mu_{n_i})^2}{2\sigma_{n_i}^2}}\right)$ where μ_{n_i} and σ_{n_i} are the mean and the deviation of the sth variable/dimension for tuples classified in the local execution class. Through calculations, we get the final equation

as presented by the Lemma. □

Based on the probability indicated by the Lemma 1, we can easily calculate the short term expected load $E(l)$ for each node just after the initial allocation of the incoming tasks. $E(l)$ can be extracted by the Binomial distribution corresponding to the aforementioned Bernoulli trial when having $|Q|$ tasks for local execution.

Lemma 2. The short term expected load for each node in the network when the Gaussian distribution is assumed is

$E(l) = \frac{\lambda}{|\mathcal{N}|}\left(1 - \frac{\pi_1}{|\mathcal{N}|\pi_2}\right)^k \sum_{m=\lfloor \frac{k}{2}+1 \rfloor}^{k} \binom{k}{m} \left(\frac{\pi_1}{|\mathcal{N}|\pi_2 - \pi_1}\right)^m$ where $\pi_1 = \prod_{\forall s} \frac{1}{\sigma_{n_i}} e^{-\frac{(y_s - \mu_{n_i})^2}{2\sigma_{n_i}^2}}$ and $\pi_2 = \prod_{\forall s} \frac{1}{\sigma_s} e^{-\frac{(y_s - \mu_s)^2}{2\sigma_s^2}}$.

Proof: The short term expected load of a node is the sum of the expected number of tasks multiplied with the probability of locally executing a task. From the Proposition 1, we have that the expected number of tasks that will be reported to n_i is $\frac{\lambda}{|\mathcal{N}|}$. For any Binomial distribution, the expected value is derived by: $\sum_{i=1}^{|Q|} i\binom{|Q|}{i} p^i (1-p)^{|Q|-i}$, where $|Q|$ is the number of the tasks reported locally. Based on the Binomial theorem, we can easily conclude that the expected value of the Binomial is $|Q|p$ (we omit the proof as it is widely studied). Based on the Lemma 1 and through substitutions, we can easily derive $E(l)$ as depicted by the Lemma 2. □

For providing a complete analysis of the problem, we also focus on the adoption of the Uniform distribution to depict the value of each attribute.

Lemma 3. The probability of locally executing T_{it} when the Uniform distribution is adopted is $p = \left(\frac{|\mathcal{N}|-1}{|\mathcal{N}|}\right)^k \sum_{m=\lfloor \frac{k}{2}+1 \rfloor}^{k} \binom{k}{m} \frac{1}{(|\mathcal{N}|-1)^m}$

Proof: For calculating p, we follow the same approach as in Lemma 1. However, the probability density function is constant in the Uniform distribution case. Through the same calculations as in Lemma 1, we get the final equation as presented by the current Lemma. □

Based on the probability indicated by Lemma 3, we can easily calculate the short term expected load $E(l)$ when the Uniform distribution is adopted.

Lemma 4. The short term expected load for each node in the network when the Uniform distribution is adopted is $E(l) = \frac{\lambda(|\mathcal{N}|-1)^k}{|\mathcal{N}|^{k+1}} \sum_{m=\lfloor \frac{k}{2}+1 \rfloor}^{k} \binom{k}{m} \frac{1}{(|\mathcal{N}|-1)^m}$.

Proof: For proving the Lemma, we adopt the same approach as in Lemma 2, however, the probability of local execution is delivered by Lemma 3. Hence, through substitutions, we can easily derive $E(l)$ as depicted by Lemma 4. □

5.2. The Long Term Expected Load

The long term expected load for each node is concluded through the transfer of tasks in the network. It consists of the load in an 'execution era' after multiple tasks rejections and transfers. In our model, we consider that after Ω decisions/transfers, every node assigns the 'rejected' tasks to the Fog/Cloud. Ω is a parameter with a strategic meaning; a high value will lead the 'rejected' tasks to 'circulate' among nodes till their final execution. In this case, tasks are transferred between nodes till they 'find' a node that will undertake the responsibility of their execution. This could happen when nodes characteristics indicate that the load and the resources (are dynamically updated) can support the action a_1. When $\Omega \to \infty$, the proposed model forces the tasks to be circulated to the network till a node decides their execution, thus, the 'execution era' is implicitly extended. However, in that case, the time required for the transfers and the communication overhead should be compared with the delay/latency realized when tasks' execution is concluded in the Fog/Cloud. In our model, Ω is defined based on the average latency for executing tasks in the Fog/Cloud and considering a minimum time for realizing a task transfer form a node to another. For instance, simulations in [27] show that the average latency for executing tasks in the Cloud is over 24 milliseconds when the tasks arrival rate increases. A low Ω indicates a limited number of 'hops' till tasks' transfer to the Fog/Cloud. Our future research plans is to define an intelligent model for delivering the final Ω.

Recall that with probability p, each node decides to locally execute a task, thus, with probability $1-p$ a task is 'rejected'. After receiving $\frac{\lambda}{|\mathcal{N}|}$ tasks, n_i will accept (i.e., locally execute) $\frac{\lambda}{|\mathcal{N}|}p$ tasks and will transfer $\frac{\lambda}{|\mathcal{N}|}(1-p)$ tasks to the network. In the 'worst' case, after the first decision, all the 'rejected' tasks could be transferred to the same peer; the same could be true for the remaining nodes. In this case, a single node, e.g., $n_{selected}$, in the entire network, will host $\frac{\lambda}{|\mathcal{N}|}\sum_{\forall i, i\neq j}(1-p_i)$ while the long term expected load of the remaining nodes will be limited.

Without loss of generality, let us focus on the 'average' case. In the average case, n_i, after every decision, uniformly distributes the 'rejected' tasks to the available $|\mathcal{N}|-1$ peers. This means that n_i will distribute $\frac{\lambda}{|\mathcal{N}|(|\mathcal{N}|-1)}(1-p_i)$ tasks to each of its peers. However, when n_i distributes the 'rejected' tasks, at the same time, the remaining nodes send their 'rejected' tasks into the network as well. Hence, each node *distributes* and *receives* a number of tasks that are 'rejected' after every decision made at the round

$\omega \in \{1, 2, 3, \ldots, \Omega\}$. It should be noted that during the distribution of tasks, nodes continue to execute the 'accepted' tasks, thus, their characteristics are updated (e.g., their load).

Lemma 5. The long term expected load for each node, at the ω decision round, is a function of the success probabilities $p_i, i = 1, 2, \ldots, |\mathcal{N}|$ calculated for every node in the network based on their characteristics.

Proof: In Table 1, we present the load for two decision rounds. In general, n_i at ω will receive $\frac{\lambda}{|\mathcal{N}|(|\mathcal{N}|-1)^{\omega-1}} \left(\sum_{j=1,j\neq i}^{|\mathcal{N}|}(1-p_j)\right)^{\omega-1}, \omega \in \{1, 2, \ldots, \Omega\}$. From those tasks, the number of the locally executed will be $\frac{\lambda}{|\mathcal{N}|(|\mathcal{N}|-1)^{\omega-1}} \left(\sum_{j=1,j\neq i}^{|\mathcal{N}|}(1-p_j)\right)^{\omega-1} p_i$ which is a function of the success probabilities calculated for each node in the network.

Table 1: Incoming and Outgoing Tasks for three decision steps.

	Tasks		
ω	Incoming	Locally Executed	Outgoing
1	$\frac{\lambda}{|\mathcal{N}|}$	$\frac{\lambda}{|\mathcal{N}|}p_i$	$\frac{\lambda}{|\mathcal{N}|}(1-p_i)$
2	$\frac{\lambda}{|\mathcal{N}|(|\mathcal{N}|-1)}\sum_{j=1,j\neq i}^{|\mathcal{N}|}(1-p_j)$	$\frac{\lambda}{|\mathcal{N}|(|\mathcal{N}|-1)}\sum_{j=1,j\neq i}^{|\mathcal{N}|}(1-p_j)p_i$	$\frac{\lambda}{|\mathcal{N}|(|\mathcal{N}|-1)}\sum_{j=1,j\neq i}^{|\mathcal{N}|}(1-p_j)(1-p_i)$

□

Based on Lemmas 2 and 5, we can easily calculate the final long term expected load for each node for an 'execution era'. We consider that the next era will start only after the final execution for every incoming task (i.e., $\Omega \to \infty$).

Lemma 6. The long term expected load for a node n_i is $E(l) = \frac{\lambda(|\mathcal{N}|-1)}{|\mathcal{N}|(|\mathcal{N}|-1-\chi)} \cdot p_i$ where $\chi = \sum_{j=1,j\neq i}^{|\mathcal{N}|}(1-p_j)$.

Proof: The long term expected load will be calculated based on the Lemma 5 for the entire set of values of ω. We consider that $\Omega \to \infty$. Hence, we should calculate the sum of the number of tasks multiplied by the probability of success for $\omega = 1, 2, \ldots, \Omega$. Based on the Lemma 5, we get that the expected number of tasks that will be locally executed is: $\frac{\lambda}{|\mathcal{N}|}p_i + \frac{\lambda}{|\mathcal{N}|(|\mathcal{N}|-1)}\left(\sum_{j=1,j\neq i}^{|\mathcal{N}|}(1-p_j)\right)p_i + \frac{\lambda}{|\mathcal{N}|(|\mathcal{N}|-1)^2}\left(\sum_{j=1,j\neq i}^{|\mathcal{N}|}(1-p_j)\right)^2 p_i + \ldots = \frac{\lambda}{|\mathcal{N}|}p_i \sum_{\omega=1}^{\Omega}\frac{\chi^{\omega-1}}{(|\mathcal{N}|-1)^{\omega-1}}$, with $\chi = \sum_{j=1,j\neq i}^{|\mathcal{N}|}(1-p_j)$. We observe that $\frac{\chi}{(|\mathcal{N}|-1)} < 1$, thus, the final long term expected load is the result of the sum of an infinite geometric series, i.e., $E(l) = \frac{\lambda(|\mathcal{N}|-1)}{|\mathcal{N}|(|\mathcal{N}|-1-\chi)} \cdot p_i$. □

An interesting observation is that if p_is for the $|\mathcal{N}-1|$ peers are equal to zero, the expected load of n_i will be infinite. This scenario depicts the above discussed worst case scenario.

6. Experimental Evaluation

In our experimental evaluation, we investigate whether the model is capable of *optimally deciding the local execution of each task or transferring this responsibility to the appropriate peer.*

6.1. Experimental Setup & Performance Metrics

In our simulations, we adopt the parameters $\langle l, r, z, b\rangle$ for decisions related to the local execution of the incoming tasks and $\langle l, r, \tau, \kappa\rangle$ for selecting the appropriate peers in the network. We focus on two aspects: **(a)** The correct identification of tasks that will be executed locally, and **(b)** The correct identification of the appropriate peer to execute the task. For the first aspect, we adopt the widely known metrics *precision* ϵ and *recall* ζ. We also adopt the *F-measure* ϕ and the *accuracy* ψ metrics. These metrics are defined as: $\epsilon = \frac{TP}{TP+FP}, \zeta = \frac{TP}{TP+FN}, \phi = 2\frac{\epsilon\zeta}{\epsilon+\zeta}, \psi = \frac{TP+TN}{TP+TN+FP+FN}$ where T refers to 'true', P refers to 'positive', F refers to 'false', and N refers to 'negative'. Hence, TP refers to true positive events, i.e., identified events that had to be identified, FP refers to false positive events, i.e., identified events that had to not been identified, and so on.

We also evaluate the selection of the appropriate peer when a task should be transferred to the group/neighborhood. We adopt random values for each parameter (i.e., $\langle l, r, \tau, \kappa\rangle$). With probability 0.20, a new message arrives to nodes indicating updates on the status of the remaining peers (i.e., load, speed, communication cost). For evaluating the selection of the appropriate peer, we focus on the mean of each parameter (i.e., $\langle l, r, \tau, \kappa\rangle$) over the selected peers. For l and κ, we target on a low mean while for the remaining parameters, we expect to observe high values. The mean is represented by the indication A in every metric i.e., $\langle l_A, r_A, \tau_A, \kappa_A\rangle$. In addition, we define metrics for identifying if the selected peers are the best among the available nodes (optimality of the model). For this, we adopt the indication B in each metric, i.e., $\langle l_B, r_B, \tau_B, \kappa_B\rangle$. The indication B in the aforementioned parameters is used to depict the difference of the value achieved by our model compared to the optimal value observed in peers' characteristics. We calculate the optimal value (the highest or the lowest depending on the parameter)

between all the nodes in the network and compare it with the characteristics of the selected node. It should be noted that a node exhibiting e.g., the lowest load, it does not mean that the same node exhibits e.g., the lowest communication cost. As our model aims to get decisions based on multiple attributes, our evaluation process manages the entire set of parameters at the same time. Any negative result means that the selected node, by our model, exhibits better characteristics than the peer with the lowest load. Based on these metrics, we aim to reveal if the proposed model is capable of selecting the best possible peer to host the 'rejected' tasks.

We adopt a simulator created in Java and train the proposed decision making mechanism adopting synthetic and real data. The real dataset is adopted from [4] and concerns data related to fire detection and the calculation of the affected area. From these data, we adopt the temperature, the humidity, the wind and the rain indication for feeding values to our parameters. We assume that the indication of fire and the presence of an affected area corresponds to action a_1, i.e., the local execution of a task. Usually, the indication of a fire is characterized by a low humidity, a high temperature and a high wind [1]. In our scenario, the decision for the local execution of a task is supported by a high priority, a low load and a high availability of resources (the complexity is combined with the remaining parameters for the final decision making). Hence, adopting the real dataset and adapting it into our scenario, we select the rain indication to 'virtually' correspond to z (we consider three priorities), the humidity to l (we target to a low load to locally execute a task like a low humidity may support the indication of a fire), the temperature to r (we target to high resources availability like a high temperature may support the indication of a fire) and the wind to b. Each tuple in the training dataset (either the synthetic or the real) is related to $\langle l, r, z, b \rangle$. The training dataset, provides various combinations of the aforementioned parameters accompanied by the appropriate decision (i.e., the corresponding action). When the tuple is classified as 1, it means that the corresponding task should be executed locally (action a_1) while a classification value 0 indicates the decision of transferring the task to the group or Fog/Cloud (actions a_2 and a_3). In the real dataset, we add the class 1 when the temperature, the humidity, the wind and the rain indicate a fire; otherwise, we add the class 0. We also present experimental results

[1] http://learningcenter.firewise.org/Firefighter-Safety/1-6.php

for the performance of our model concerning the expected load of each node. We perform the evaluation of our scheme for $|\mathcal{N}| = \{10, 50, 100, 1000\}$. We study how $|\mathcal{N}|$ affects the results. In addition, we consider four (4) experimental scenarios: (i) Scenario A: $\{w_s\} = \{0.3, 0.3, 0.3, 0.1\}$; (ii) Scenario B: $\{w_s\} = \{0.6, 0.2, 0.1, 0.1\}$; (iii) Scenario C: $\{w_s\} = \{0.2, 0.2, 0.4, 0.4\}$; (iv) Scenario D: $\{w_s\} = \{0.2, 0.1, 0.1, 0.6\}$. Through the aforementioned scenarios, we pay attention on different parameters when selecting a node for transferring the task. For instance, Scenario A pays equal attention on the load, the resources and the speed while Scenario B pays more attention on the load. Scenario C pays more attention on the speed and the cost and, finally, Scenario D pays attention on the cost.

6.2. Performance Evaluation

Initially, we report on the complexity of the proposed model. This complexity is affected by: (i) the complexity of the kNNC; (ii) the complexity of the utilities calculation; (iii) the complexity of the utilities aggregation; (iv) the complexity of the sorting process to produce the final utilities list. In the worst case, the complexity for (i) is $\mathrm{O}(k|D|+|D|Y)$, where D is the training dataset, k is the number of neighbors and Y is the number of dimensions/variables. In Fig. 3, we present the plot of the discussed complexity. In addition, the complexity for (ii) and (iii) is $\mathrm{O}(|\mathcal{N}|)$ while the complexity for (iv) is $\mathrm{O}(|\mathcal{N}|log|\mathcal{N}|)$ (we can rely on a fast sorting technique e.g., merge or heap sort). In Fig. 4, we keep Y constant and present the complexity of the proposed scheme for various numbers of the length of the training dataset and the number of the peers. Based on the above, the final complexity in the worst case scenario is $\mathrm{O}(k|D| + |D|Y + |\mathcal{N}|log|\mathcal{N}|)$. We compare the complexity of our model with other relevant efforts in the domain. In [53], the authors discuss three task scheduling algorithms, the EASU (Energy-Aware Scheduling under Uncertainty), the RAS (Reliability-Aware Scheduling) and the basic resource provisioning algorithm. The complexity of the EASU is $O(|T||VMs||\mathcal{N}|)$ where $|VMs|$ is the number of the VMs where the tasks should be allocated and $|\mathcal{N}|$ is the number of hosts like in our case. The complexity of the RAS is $O(|VMs||\mathcal{N}||\mathcal{N}|_C)$ with $|\mathcal{N}|_C$ depicting the number of candidate hosts. In [35], the tasks allocation problem is seen as a capacity problem. The network is modeled as a graph like in our case. The complexity of the proposed algorithm is $O(|\mathcal{N}| + (|\mathcal{N}|^2|T|) + |\mathcal{N}||E|^2)$ (for the calculation of the last part of the complexity we consider the Edmonds-Karp algorithm for finding the maximum flow in a graph). Finally, in [56],

the authors solve the task allocation problem through the adoption of the network flow method and conclude that the intranode communication cost is a key to the complexity of the algorithm. They prove that complexity is $O(|\mathcal{N}|^2|T|^4)$ if the intranode communication cost equals the internode communication cost. In addition, the convex cost flow version of the algorithm can be solved in $O(|\mathcal{N}|^2|T|^2 log(|\mathcal{N}| + |T|))$.

In the following experiments, we try to reveal the short term expected load for each node when we adopt various realizations for parameters λ, $|\mathcal{N}|$ and Y. Our aim is to see how many tasks will be hosted locally when $\omega = 1$. For this, we simulate the arrival of 1,000 TRTs and for each one, we adopt the Uniform and the Gaussian distributions to get values for every parameter participating in the envisioned tuples. The Uniform distribution is adopted to simulate a very dynamic environment where parameters can change values allocated in the entire interval $[0, 1]$. On the other hand, the Gaussian distribution is adopted to simulate a more 'stable' environment where parameters realizations are allocated around the mean. In Figure 5, we present our results for the expected load $E(l)$ and for different λ. As natural, we observe that an increment in the number of the incoming tasks in the entire network will increase the short term expected load of nodes. The interesting is that the Uniform distribution results more tasks for local execution compared to the Gaussian distribution. The reason is that the local datasets, in the Uniform distribution case, contain data that are not concentrated around the mean increasing the probability for local execution. Recall that the probability for having a task locally executed depends on the 'similarity' between the characteristics of every task and the available dataset adopted for delivering the kNNs, thus, the final decision. Another interesting observation is as follows. In this set of experiments, we get $|\mathcal{N}| = 5$ while $E(l) = \{46, 33\}$ for the Uniform and the Gaussian distributions, respectively. If we consider that 1,000 tasks arrive in the network in a time unit, thus, approximately, 200 tasks are reported at each node, only 230 and 165 tasks, respectively, will be executed locally after the first decision while the remaining will be transferred to peers or the Fog/Cloud. It should be noted that those numbers do not include the tasks that are transferred in the current node by peers.

In Figure 6, we show our results for short term $E(l)$ and for different $|\mathcal{N}|$. In these plots, we do not present the results for $|\mathcal{N}| = 1$ because it is the only result above unity. For all the remaining experimental scenarios, we get the short term $E(l)$ below unity, which indicates that the load for each node

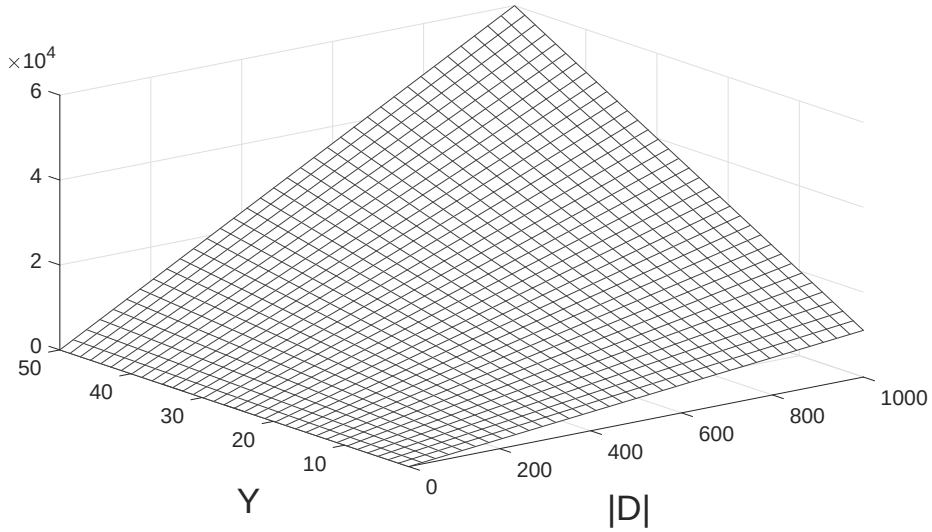


Figure 3: The complexity of the kNNC classifier.

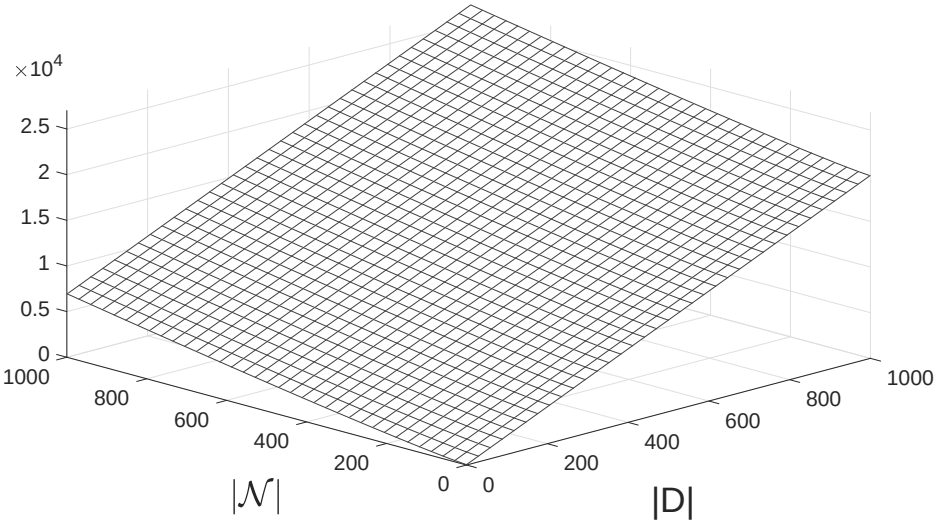
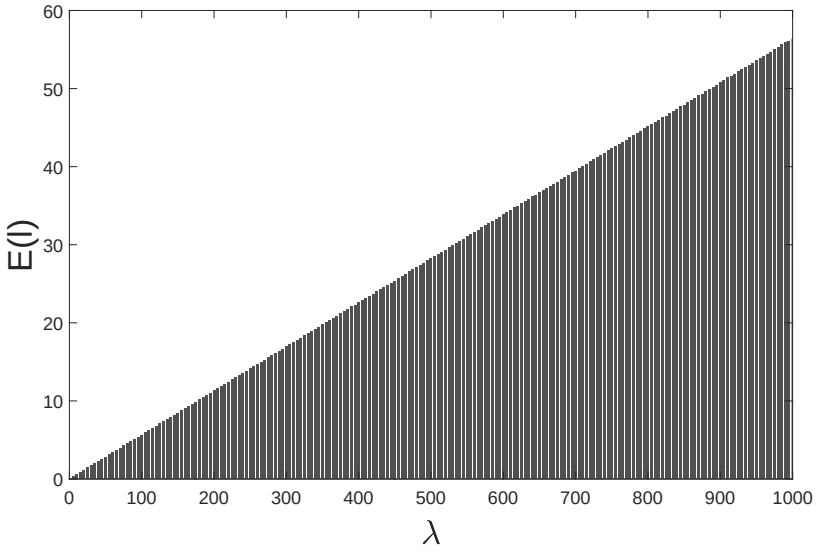
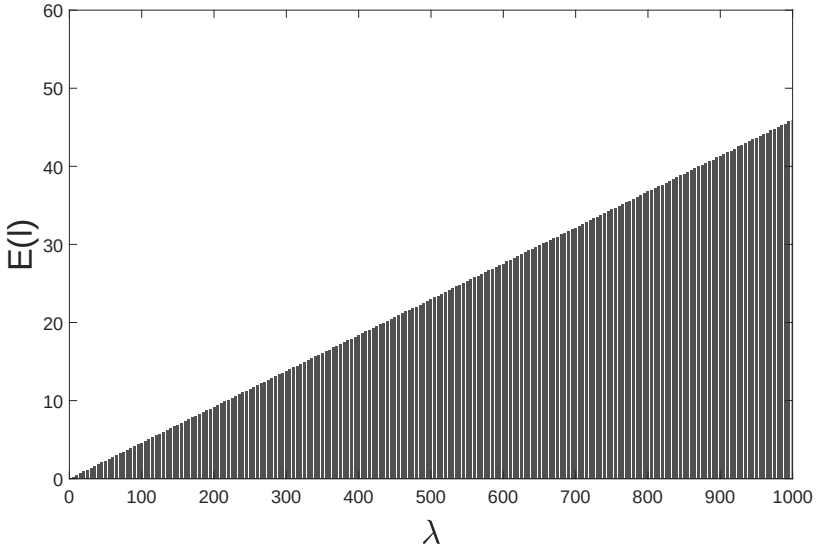


Figure 4: The complexity of the proposed model.



(a) Uniform distribution.



(b) Gaussian distribution.

Figure 5: Expected load for various λ values.

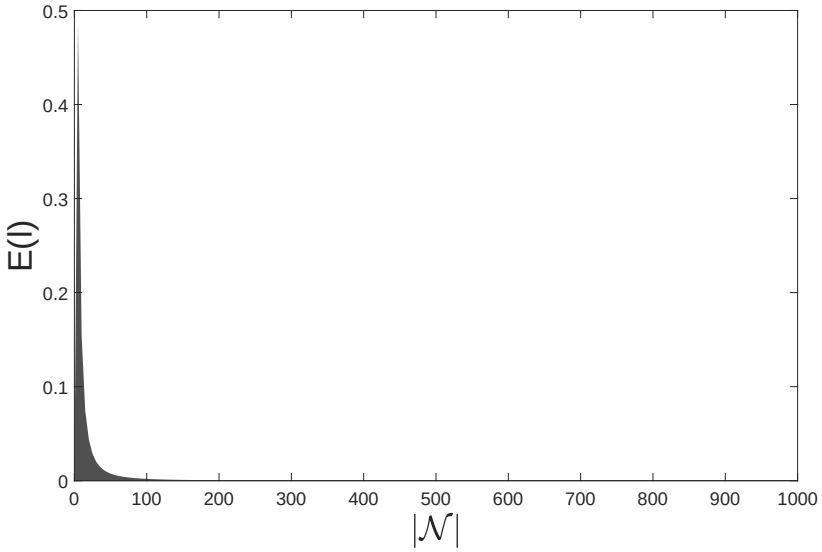
is limited. This is more intense when $|\mathcal{N}| \to 1,000$, thus, tasks are offloaded

to peer nodes instead of being kept locally. This could lead to a situation where nodes could execute tasks reported to other nodes instead of keeping 'their' tasks. This is natural as nodes want to offload the incoming tasks and keep only tasks that are important to be locally executed as indicated by the adopted classifier. However, such an approach may lead to an exchange of tasks in the long term, thus, tasks will be continuously circulated in the network.

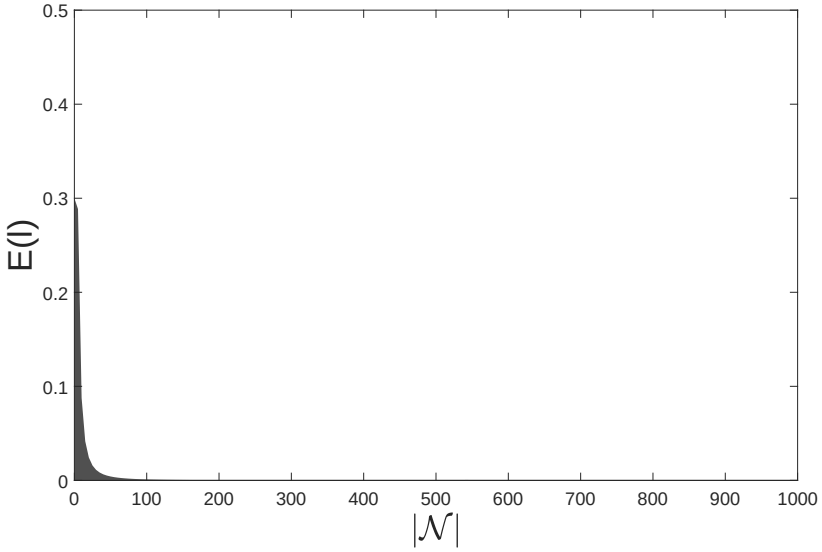
The performance of the proposed model for various Y realizations, is depicted in Figure 7. These results are similar as in the previous experimental scenarios, i.e., the adoption of the Uniform distribution leads to a high $E(l)$. In addition, our results exhibit that when $Y \leq 10$, fluctuations in the expected load can be present. However, such fluctuations are eliminated and the proposed model exhibits 'stability' as $Y \rightarrow 1,000$. Again, the adoption of the Uniform distribution leads to higher expected load compared to the scenario where the Gaussian distribution is adopted to produce values for each parameter. Again the observed outcomes can be considered as natural as the production of values based on the Uniform distribution depicts a very dynamic environment where nodes' and tasks' characteristics are continuously updated. Hence, when a task is generated and reported to a node, peers' characteristics may be completely different compared to the previous execution round. For instance, the Uniform distribution can 'generate' a low value at timestep t while at $t+1$ a high value can be the case. This way, it is natural to observe higher values for the load compared to the scenario where the Gaussian distribution is adopted. The reason is that in the Uniform distribution scenario, the load is not gathered around the mean but it is produced in the entire interval in consecutive decision rounds.

In Figure 8, we plot the nodes' expected load vs the rate λ and $|\mathcal{N}|$. In this set of simulations, we experiment with $p \in \{0.2, 0.8\}$. When $p = 0.2$, each node will locally execute the 20% of the incoming tasks while $p = 0.8$ indicates that nodes will locally execute the 80% of the incoming tasks. The expected load 'follows' the probability p, i.e., a low p leads to a low expected load. This is natural, however, it is interesting to observe that, no matter p, when the number of nodes is high, the expected load for each node will be minimized. These outcomes confirm our expectations for the load of the nodes when tasks arrive at high rates.

In Figure 9, we present our results for the long term expected for each node. In this set of experiments, we consider that the success probability for peers is around 0.5. We observe that the expected load, in the lifetime



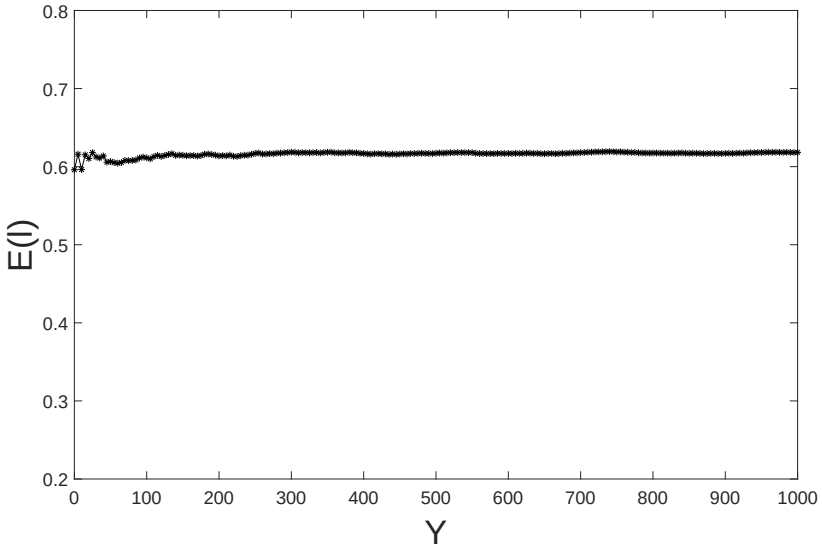
(a) Uniform distribution.



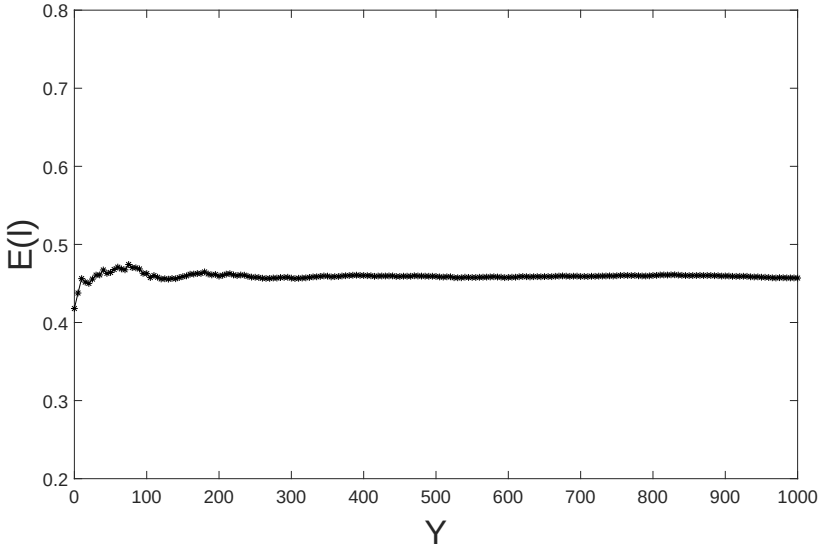
(b) Gaussian distribution.

Figure 6: Short term expected load for various $|\mathcal{N}|$ values.

of the network, is low except the scenarios where the number of nodes is



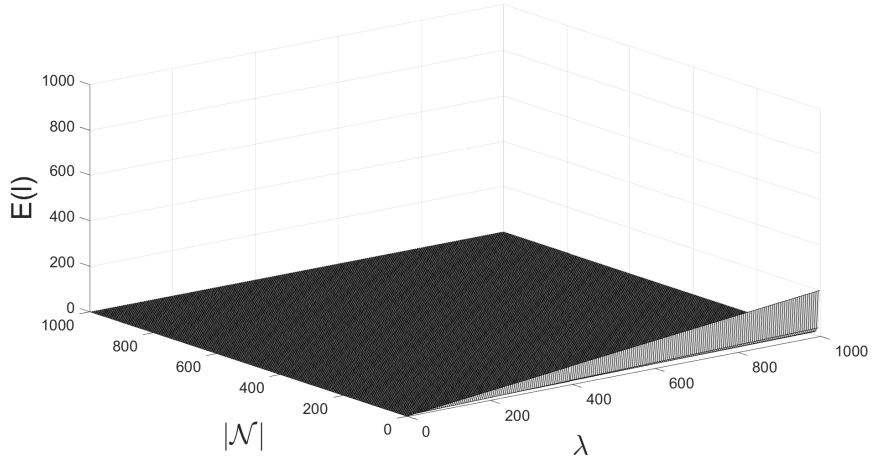
(a) Uniform distribution.



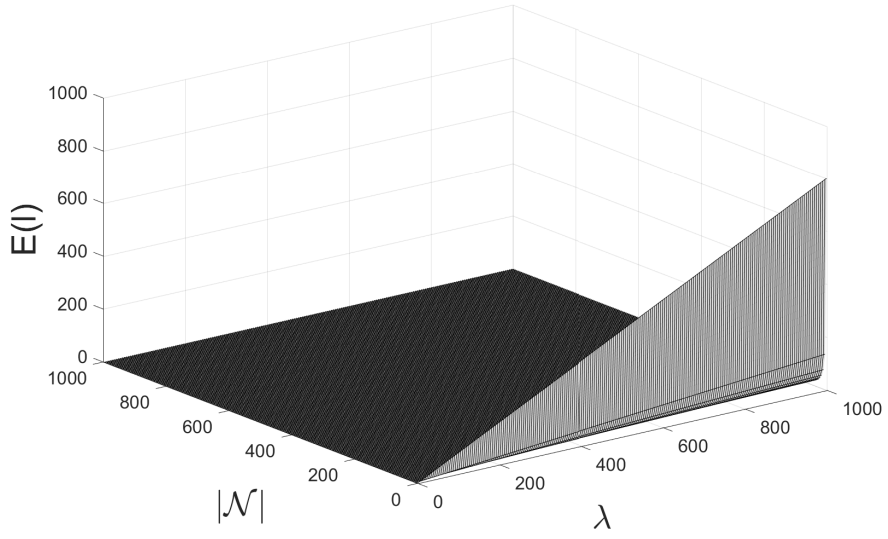
(b) Gaussian distribution.

Figure 7: Short term expected load vs different TRT cardinality.

limited and the local probability of success is high (close to unity). These



(a) p=0.2.



(b) p=0.8.

Figure 8: Short term expected load vs the probability of local execution.

experiments enhance our view that when $|\mathcal{N}|$ is high, the incoming tasks could be served, in total, by the network without the need of transferring tasks to the Fog/Cloud. When we have a high number of nodes present in the network, we secure the execution of the incoming tasks no matter their

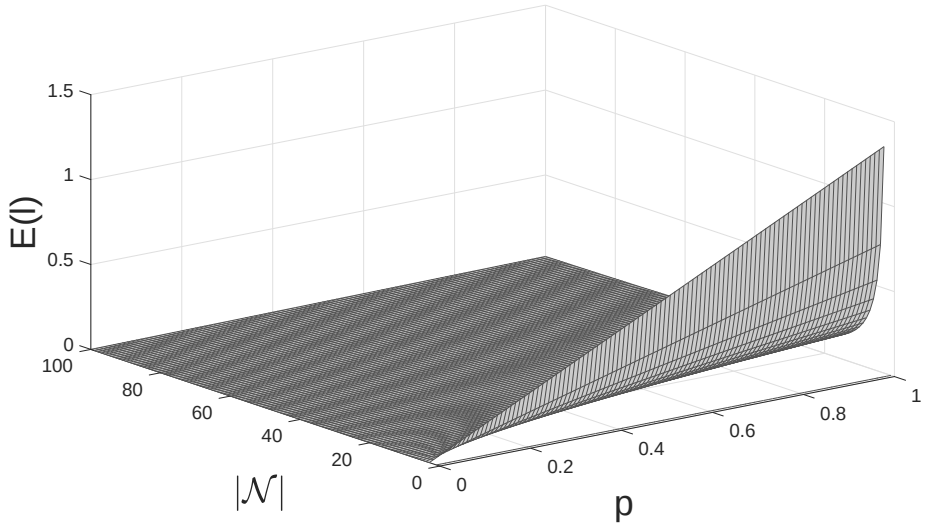


Figure 9: Long term expected load vs p and $|\mathcal{N}|$.

characteristics while the load of each node remains at low levels. However, as already mentioned, we should take into consideration the burden for transferring tasks in the network compared to the latency that we will enjoy if tasks should be allocated to the Fog/Cloud. In addition, these results depict the load for receiving tasks in a time unit (e.g., second, hour, day, week), i.e., an 'execution era'. This means that we can easily support multiple groups of tasks, thus, multiple 'execution eras' as the load will be also limited.

In our simulations, the kNNC exhibits high performance as it results $\epsilon = 1.0$. This means that no false positive events are identified, thus, the corresponding tasks are correctly transferred to the network. In addition, we observe $\zeta = 0.97$ which means that false negatives events are also limited. For the remaining metrics, we get $\phi = 0.98$ and $\psi = 0.98$. These results expose the accuracy of our model in the identification of the tasks that should be locally executed.

We also evaluate our scheme in the peer selection process. We want to identify if the characteristics of the selected peer are those that facilitate the execution of the allocated task. For this, we deliver the mean values for each parameter calculated over a high number of experiments. In Fig. 10, we present our results for l and for each experimental scenario. We observe that the lowest load is achieved in Scenario B. In this scenario, the weight of the

utility for load is the highest among the available weights, thus, the proposed scheme naturally pays more attention on the load of the peer where every task will be allocated. Recall that weights are adopted in the calculation of the utility in the additive utility function. We also observe that the higher the number of nodes, the lower the load of the selected peer becomes confirming again the above discussed outcomes. On the other hand, in Scenarios C & D, we observe the highest average load, i.e., in scenarios where l is assigned with a low weight compared to the remaining parameters. The mean l is below 0.4 when $|\mathcal{N}| \rightarrow 1,000$. The increased number of nodes positively affects the final outcome as our scheme has many alternatives for selecting the final peer. The involvement of multiple utility functions (an ensemble scheme) manages to find the appropriate peers when a task should be allocated in the network.

In Fig. 11, we present our results concerning the available resources r in the selected peer. In the majority of the results r is above 0.5 while the increased number of nodes assists in the increment of the final outcome. The highest mean r is observed in Scenarios B and C when $|\mathcal{N}| = 1,000$. Our outcomes indicate that the selected nodes are characterized by a high availability of resources that can be devoted to the execution of the allocated tasks. In Fig. 12, we observe our results concerning τ. We get the highest outcome in Scenario A when $|\mathcal{N}| = 100$. In general, the discussed results are judged as efficient due to that τ is close or over 0.6 for the majority of the cases. Our experimental evaluation for κ is presented in Fig. 13. We observe that the lowest value is related to Scenario D where we assign the highest weight for κ. Outcomes show that the increased weight and the increased number of nodes assist our scheme to select a peer with a low communication cost. One can observe that, in the majority of the experimental scenarios, κ is below 0.4 while it is approaching 0.2 when $|\mathcal{N}| \rightarrow 1,000$. The high number of nodes present in the network gives more opportunities to find the lowest possible communication cost, thus, it minimizes the communication overhead of the network. This way, we can limit the resources and the time required to allocate tasks and get the final response.

We devote a set of experiments to reveal the allocation of the 'rejected' tasks to the appropriate peer. We focus on the multi-attribute optimality, meaning that the proposed scheme takes into consideration all the parameters at the same time. In Table 2, we present our results for the Scenarios A and B. We observe that the difference of the selected peer (as resulted by our model) with the peer exhibiting the lowest load is around 0.420 and 0.270

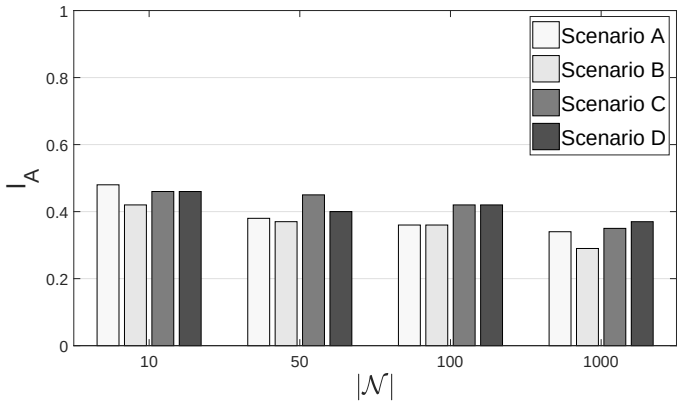


Figure 10: Our results for the mean load of the selected peers.

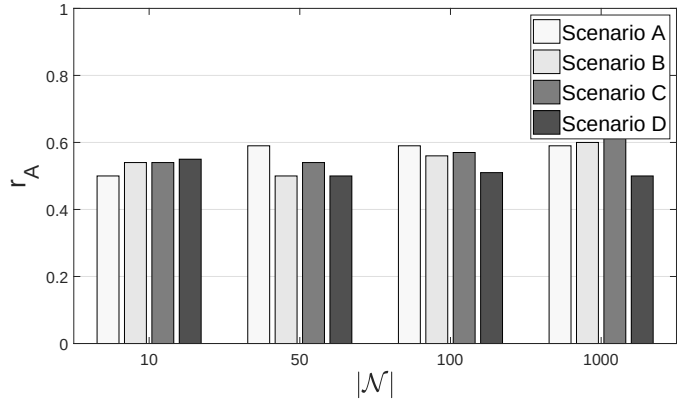


Figure 11: Our results for the mean resources of the selected peers.

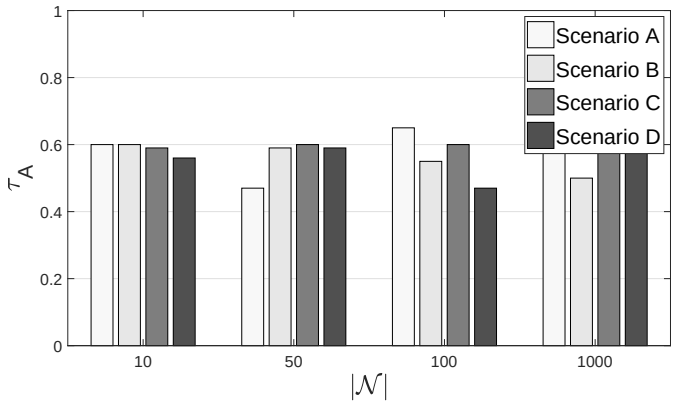


Figure 12: Our results for the mean speed of the selected peers.

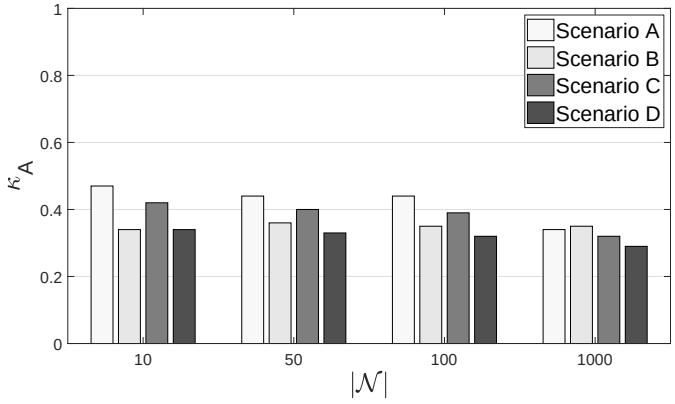


Figure 13: Our results for the mean cost of the selected peers.

for Scenarios A and B, respectively. Recall that in Scenario B, the weight of the load is the highest when calculating the final utility. We observe that the increased weight leads to the selection of a peer that is close to the optimal decision. In any case, decisions under the Scenario B lead to better results for l compared to the decisions made under the Scenario A. Our model manages to select a peer that exhibits 'better' characteristics for the remaining parameters in the majority of the experimental scenarios. This is depicted by the negative values, i.e., the selected peer exhibits higher resources, speed and lower cost than the peer with the lowest load in the network. Similar results we get if we focus on Scenarios C and D (see Table 3). These outcomes support the observation that our model can be adopted to select peers exhibiting the best possible characteristics under the perspective of having a multi-attribute decision making.

Table 2: Optimality results for Scenarios A & B (synthetic trace).

	Scenario A				Scenario B			
$\|\mathcal{N}\|$	l_B	r_B	τ_B	κ_B	l_B	r_B	τ_B	κ_B
10	0.420	-0.125	-0.123	-0.112	0.270	-0.036	-0.016	-0.120
50	0.350	-0.120	0.060	-0.220	0.310	-0.220	-0.150	0.004
100	0.380	-0.090	-0.200	-0.106	0.290	0.090	-0.230	-0.080
1,000	0.310	-0.020	-0.040	-0.030	0.290	-0.059	-0.096	-0.370

Table 3: Optimality results for Scenarios C & D (synthetic trace).

	Scenario C				Scenario D			
$\|\mathcal{N}\|$	l_B	r_B	τ_B	κ_B	l_B	r_B	τ_B	κ_B
10	0.290	0.029	-0.010	-0.109	0.350	0.040	0.120	-0.140
50	0.350	-0.114	-0.010	-0.196	0.370	-0.100	0.009	-0.001
100	0.420	-0.128	-0.001	-0.146	0.330	-0.085	-0.233	0.036
1,000	0.310	-0.060	-0.138	-0.136	0.330	-0.103	-0.045	-0.162

We also perform a set of experiments adopting the real dataset. Our results deliver $\epsilon = 0.81$, $\zeta = 0.49$, $\psi = 0.67$, and $\phi = 0.61$. We observe that, compared to the synthetic trace, there is an increased number of false positives and false negatives. A number of positive events (i.e., the local

execution of a task) is not identified, thus, the corresponding tasks are transferred to peers. From the 517 tasks, 163 are locally executed while 354 are transferred to peer nodes. The interesting is that no task is transferred to be executed in Fog/Cloud.

We report on the optimality results for the real dataset. Tables 4 and 5 present our outcomes for Scenario A, Scenario B, Scenario C and Scenario D, respectively. In general, the load of the selected nodes is lower than in the experiments realized with the synthetic trace. The proposed model manages to select nodes that their characteristics are close or lower than the best possible node selection. The interesting is that in this set of experiments, the number of the nodes present in the network affects our results. One can observe an increment for l or κ as $|\mathcal{N}| \rightarrow 1,000$. For Scenario A, the real trace exhibits the best performance for l and κ while for the remaining metrics the best performance is achieved through the adoption of the synthetic trace. For Scenario B, we observe similar performance in both cases with some fluctuations in the results. For Scenario C, the adoption of the real dataset leads to the best performance for l while the adoption of the synthetic trace leads to the best performance for κ. Similar performance is observed for r and τ. Finally, for Scenario D, the synthetic trace leads to the best results for τ while for the remaining metrics we observe a similar behavior.

Table 4: Optimality results for Scenarios A & B (real dataset).

	Scenario A				Scenario B			
$\|\mathcal{N}\|$	l_B	r_B	τ_B	κ_B	l_B	r_B	τ_B	κ_B
10	0.205	-0.055	0.091	-0.055	0.174	0.009	0.107	-0.009
50	0.264	0.004	0.043	-0.004	0.209	0.080	-0.065	-0.080
100	0.264	-0.082	-0.088	0.082	0.261	0.020	-0.053	-0.020
1,000	0.287	-0.310	0.004	0.310	0.293	-0.296	0.010	0.296

Table 5: Optimality results for Scenarios C & D (real dataset).

	Scenario C				Scenario D			
$\|\mathcal{N}\|$	l_B	r_B	τ_B	κ_B	l_B	r_B	τ_B	κ_B
10	0.193	0.084	0.031	-0.084	0.194	0.041	0.079	-0.041
50	0.270	-0.056	0.031	0.056	0.262	0.067	-0.050	-0.067
100	0.257	-0.029	-0.035	0.029	0.272	0.033	0.009	-0.033
1,000	0.269	-0.293	0.007	0.293	0.277	-0.295	0.019	0.295

We compare the performance of our model (i.e., the 'Model') with the scheme presented in [3]. In [3], the authors propose a task scheduling algorithm (ETSI) for IoT that is based on a heuristic to finalize the allocations of tasks to the available nodes. The algorithm adopts a node state analyzer that delivers the final outcome based on the remaining energy, the distance from the edge of the network and the number of neighbors. The edge gateway calculates the rank of each node and decides on the final allocation for each task. Actually, the node with the lowest ranking is selected for the final allocation. In Tables 6, and 7, we present our results for Scenario A, Scenario B, Scenario C and Scenario D, respectively. As the experimental results presented in [3] focus on the network lifetime and load, in our results, we consider the outcomes for l_B and r_B. We observe that the Model outperforms the ETSI for all the experimental scenarios. There is a significant difference in the load of the selected node where tasks will be allocated. The reason is that ETSI mainly takes into consideration the energy issues and the distance of nodes from the gateway without paying significant attention on the load. However, the increased load may negatively affect the energy resources especially when the allocated tasks are characterized by an increased complexity.

Table 6: Comparison of the optimality results (Scenarios A & B).

	Scenario A				Scenario B			
	Model		ETSI		Model		ETSI	
$\|\mathcal{N}\|$	l_B	r_B	l_B	r_B	l_B	r_B	l_B	r_B
100	0.264	-0.082	0.490	0.058	0.261	0.020	0.540	0.017
1,000	0.287	-0.310	0.701	-0.057	0.293	-0.296	0.687	-0.042

Table 7: Comparison of the optimality results (Scenarios C & D).

	Scenario C				Scenario D			
	Model		ETSI		Model		ETSI	
$\|\mathcal{N}\|$	l_B	r_B	l_B	r_B	l_B	r_B	l_B	r_B
100	0.257	-0.029	0.489	0.064	0.272	0.033	0.501	0.076
1,000	0.269	-0.293	0.710	-0.044	0.277	-0.295	0.698	-0.067

Trying to reveal the hidden aspects of the performance of our model, we provide plots for the probability density estimation (pde) for each parameter in Fig. 14. We set equal weights (i.e., 0.25) for each parameter and record their realization. We observe that l and κ are concentrated in the first half of the interval $[0, 1]$ while τ is concentrated at the upper part of the same interval. r is concentrated at the middle of the interval. In any case, these results exhibit the potential of the proposed model to select the appropriate peer when it is necessary.

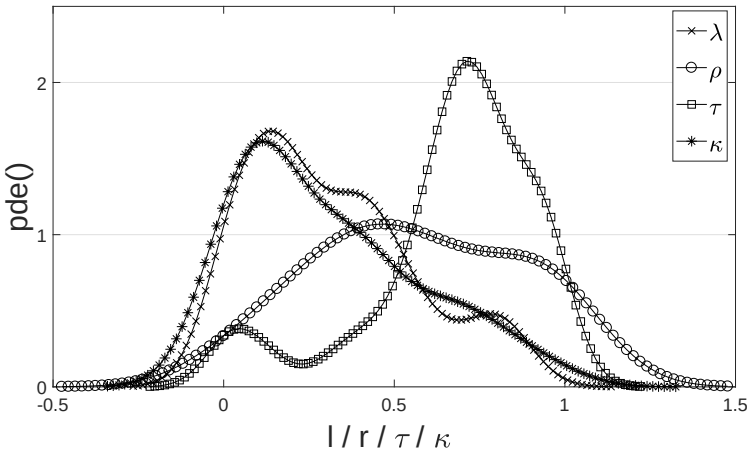


Figure 14: Probability density estimation for the model parameters.

7. Conclusions

IoT nodes are capable of collecting and processing various data becoming knowledge providers. Recent advances in the IoT domain involve the execution of processing tasks at the edge, at the nodes, to limit the time for

retrieving results. IoT nodes are characterized by limited computational capabilities while they are instructed to execute multiple tasks. Hence, nodes, when tasks 'arrive', should select the tasks that will be executed locally or be transferred to their peers in the network. This way, we create a collaborative, however, distributed scheme for tasks execution trying to find paths for minimizing the response time without jeopardizing the resources of nodes.

We describe a scheme that, in a distributed manner, sequentially decides where the tasks will be processed. Our scheme pushes the task allocation intelligence into the edge network by providing a two-level decision making mechanism. The mechanism derives decisions related to which tasks will be executed locally in the nodes. The proposed decision making scheme sequentially examines possible actions to optimally decide the place where tasks will be executed. Every node autonomously decides the execution of tasks. We evaluate our scheme with synthetic and real data and provide numerical results. We show that the proposed scheme manages to deliver decisions that optimally select the place of the execution based on a set of parameters. When the decision concerns the group of nodes, the proposed scheme selects a node in the network edge and transfers the task there. Our experimental evaluation shows that the selected nodes are appropriate minimizing the average difference with the optimal solution. Our future research plans involve the provision of an uncertainty management model. Such a model could incorporate more easily the uncertainty present in the decision making process.

Acknowledgment

This work is funded by the EU/H2020 Marie Sklodowska-Curie Action Individual Fellowship (MSCA-IF-2016) under the project INNOVATE (Grant No. 745829).

[1] Aazam, M., Hung, P. P., Huh, E. N., 'Smart Gateway based Communication for Cloud of Things', 9th ICSSNIP, 2014.

[2] Awadalla, M. H. A., 'Task Mapping and Scheduling in Wireless Sensor Networks', Int. Journal of Computer Science, 440(4), 2013.

[3] Baranidharan, B., Saravanan, K., 'ETSI: Efficient Task Allocation in Internet of Things', International Journal of Pure and Applied Mathematics, vol. 117(22), 2017, pp. 229–233.

[4] Cortez, P., Morais, A., 'A Data Mining Approach to Predict Forest Fires using Meteorological Data', In Proc. of the 13th Portuguese Conference on Artificial Intelligence, Guimares, Portugal, 2007, pp. 512–523.

[5] De Benedetti, M., Messina, F., Pappalardo, G., Santoro, C., 'JarvSis: A Distributed Scheduler for IoT Applications', Cluster Computing, 20, 2017, pp. 1775–1790.

[6] Bharti, S., Pattanaik, K., 'Task Requirement Aware Pre-processing and Scheduling for IoT Sensory Environments', Ad Hoc Networks, 50, 2016, pp. 102–114.

[7] Chen, Z., Hu, W., Wang, J., Zhao, S., Amos, B., Wu, G., Ha, K., Elgazzar, K., Pillai, P., Klatzky, R., Siewiorek, D., Satyanarayanan, M., 'An Empirical Study of Latency in an Emerging Class of Edge Computing Applications for Wearable Cognitive Assistance', in Proc. of the 2nd ACM/IEEE Symposium on Edge Computing, 2017.

[8] Choi, H., Lim, J. B.,, Yu, H., Lee, E. Y., 'Task Classification Based Energy-Aware Consolidation in Clouds', Scientific Programming, vol. 2016, 2016.

[9] Coltin, B., Veloso, N., 'Mobile Robot Task Allocation in Hybrid Wireless Sensors Networks', in 2010 Int. Conf. on Intelligent Robots and Systems, 2010.

[10] Dai, L., Chang, Y., Shen, Z., 'An Optimal Task Scheduling Algorithm in Wireless Sensor Networks', Int. JCCC, 1, 2011, pp. 101–112.

[11] Dueck, G., Scheuer, T., 'Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing', Journal of Computational Physics, vol. 90(1), 1990, pp. 161–175.

[12] Edalat, N., Xiao, W., Tham, C.-K., Keikha, E., Ong, L.-L., 'A price-based adaptive task allocation for Wireless Sensor Network', 6th IC-MASS, 2009.

[13] Emerso, P., 'The Original Borda Count and Welfare', Social Choice and Welfare, 40(2), 2013, pp. 353-358.

[14] Formato, R., 'Dynamic Threshold Optimization - A New Approach?', arXiv:1206:0414, 2012.

[15] Gaines, B., 'A Situated Classification Solution of a Resource Allocation Task Represented in a Visual Language', International Journal of Human-Computer Studies, vol. 40(2), 1994, pp. 243–271.

[16] Garcia Lopez, P., Montresor, A., Epema, D., Datta, A., Higashino, T., Iamnitchi, A., Barcellos, M., Felber, P., Riviere, E., 'Edge-centric Computing: Vision and Challenges', ACM SIGCOMM Computer Communication Review, 45(5), 2015.

[17] Garmabaki, A. H. S., Ahmadi, A., Ahmadi, M., 'Maintenance Optimization Using Multi-Attribute Utility Theory', Current Trends in Reliability, Availability, Maintainability and Safety, 2015, pp. 13–25.

[18] Gennert, M., Yuille, A., 'Determining the Optimal Weights in Multiple Objective Function Optimization', in Proc. of the 2nd International Conference on Computer Vision, 1988.

[19] Greenberg, A., Hamilton, J., Maltz, D., Patel, 'The Cost of a Cloud: Research Problems in Data Center Networks', ACM SIGCOMM, 39(1), 2008, pp. 68–73.

[20] Han, J., Kamber, M., Pei, J., 'Data Mining, Concepts and Techniques', Morgan Kaufmann Publishers, 2012.

[21] Hershenson, M., Boyd, S., Lee, T., 'GPCAD: a tool for CMOS opamp synthesis', in proc. of IEEE/ACM International Conference on Computer-Aided Design, pp. 296-303, 1998.

[22] Hu, X., Xu, B., 'Task Allocation Mechanism Based on Genetic Algorithm in Wireless Sensor Networks', in ICAIC, 2011.

[23] Keeney, R. L., Raiffa, H., 'Decisions with multiple objectives: preferences and value trade-offs', Cambridge University Press, 1993.

[24] Kolomvatsos, K., Loukopoulos, T., 'Scheduling the Execution of Tasks at the Edge', 2018 IEEE Conference on Evolving and Adaptive Intelligent Systems, Rhodes, Greece, May 25-27, 2018.

[25] Krishnapriya, S., Joby, P. P., 'QoS Aware Resource Scheduling in Internet of Things-Cloud Environment', Int. JSER, 6(4), 2015.

[26] Lam, L., 'Theory and Application of Majority Vote - From Condorcet Jury Theorem to Pattern Recognition', 2nd International Conference on mathematics education into the 21st century: Mathematics for Living, Amman, Jordan, 2000.

[27] Liu, J., Mao, Y., Zhang, J., Letaief, K. B., 'Delay-Optimal Computation Task Scheduling for Mobile-Edge Computing Systems', https://arxiv.org/abs/1604.07525, 2016.

[28] Morabito, R., 'Virtulization on Internet of Things Edge Devices with Container Technologies: A Performance Evaluation', IEEE Access, vol. 5, 2017, pp. 8835–8850.

[29] Morabito, R., Cozzolino, V., Yi Ding, A., Beijar, N., Ott, J., 'Consolidate IoT Edge Computing with Lighweight Virtualization', IEEE Network, 2018.

[30] Moschakis, I., Karatza, H., 'Towards Scheduling for Internet-of-Things Applications on Clouds: A Simulated Annealing Approach', Concurrency and Computation, Combined Special Issues on the Internet of Things, 27(8), 2015, pp. 1886–1899.

[31] Narula, S., 'Hierarchical location-allocation problems: A classification scheme', European Journal of Operational Research, vol. 15(1), 1984, 93-99.

[32] Nelson, R., 'Probability, Stochastic Processes, and Queueing Theory', Springer-Verlag, New York, 1995.

[33] Pahl, C., Helmer, S., Miori, L., Sanin, J., Lee, B., 'A Container-Based Edge Cloud PaaS Architecture Based on Raspberry Pi Clusters', in proc. of the 4th IEEE International Conference on Future Internet of Things and Cloud Workshops, Vienna, Austria, 2016.

[34] Pahl, C., Lee, B., 'Containers and Clusters for Edge Cloud Architectures - a Technology Review', in proc. of the 3rd International Conference on Future Internet of Things and Cloud, 2015.

[35] Pasteris, S., Wang, S., Makya, C., Chan, K., Herbster, M., 'Data Distribution and Scheduling for Distributed Analytics Tasks', IEEE SWC Conference, 2017.

[36] Prince, A., Smolensky, P., 'Optimality Theory: Constraint Interaction in Generative Grammar', y, 2008.

[37] Razavinegad, A., 'Task Allocation In Robot Mobile Wireless Sensor Networks', Int. Journal of Scientific & Technology Research, 3(6), 2014.

[38] Roman, R., Lopez, J., Mambo, M., 'Mobile edge computing, Fog et al.: A survey and analysis of security threats and challenges', Future Generation Systems, 78(2), 2018, pp. 680–698.

[39] Santos, J., Wauters, T., Volckaert, B., De truck, F., 'Fog Computing: Enabling the Management and Orchestration of Smart City Applications in 5G Networks', Entropy, vol. 4, 2018.

[40] Satyanarayanan, M., 'A brief history of cloud offload: A personal journey from Odyssey through cyber foraging to cloudlets', Mobile Computing Communications, 18(4), 2015, pp. 19–23.

[41] Shanthan, BJ., Kumar, A. D. V., Govindrajan, E., Arockian, L., 'Scheduling for Internet of Things Applications on Cloud: A Review', Imperial Journal of Interdisciplinary Research, 3(1), 2017.

[42] Silberschatz, A., Baer Galvin, P., Gagne, G., 'Operating Systems Concepts', 9th Ed., Wiley, 2013.

[43] Soorapanth, T., 'Understanding and Optimizing Weight Factors in Multi-Objective Geometric Programming', in proc. of the ECTI International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, 2010.

[44] Soorapanth, T., 'Gaining insights in analog design via Geometric Programming', in proc. of the International Symposium on Communications and Information Technologies, pp. 121-126, 2007.

[45] Soylu, B., Kapan Ulusoy, S., 'A preference ordered classification for a multi-objective max-min redundancy allocation problem', Computers and Operations Research, vol. 38(12), 2011, pp. 1855–1866.

[46] Sun, G., Li, Y., Liao, D., Chang, V., 'Law-Latency Orchestration for Workflow-Oriented Service Function Chain in Edge Computing', Future Generation Computer Systems, vol. 85, 2018, pp. 116–128.

[47] Sun, J., Sun, S., Li, K., Liao, D., Sangaiah, A. K., Chang, V., 'Efficient Algorithm for Traffic Engineering in Cloud-of-Things and Edge Computing', Computers and Electrical Engineering, vol. 69, 2018, pp. 610–627.

[48] Tian, Y., Ekici, E., Ozguner, F., 'Energy-Constrained Task Mapping and Scheduling in Wireless Sensor Networks', IEEE ICMASS, 2005.

[49] Varshney, P., 'Distributed Detection and Data Fusion', Springer, 1997.

[50] Voigt, T., Fried, R., Backes, M., Rhode, W., 'Threshold optimization for classification in imbalanced data in a problem of gamma-ray astronomy', Advances in Data Analysis and Classification, vol. 8(2), 2014, pp. 195–216.

[51] Voinescu, A., Tudose, D. S., Tapus, N., 'Task Scheduling in Wireless Sensor Networks', 6th ICNS, 2010.

[52] Weiss, S., Arne Schwarz, J., Stolletz, R., 'The Buffer Allocation Problem: A Joint Classification and Review of Decision Problems, Solution Approaches, and Test Instances', SSRN Electronic Journal, 10.2139/ssrn.2843435.

[53] Wu, G., Bao, W., Zhang, X., 'A General Cross-Layer Cloud Scheduling Framework for Multiple IoT Computer Tasks', Sensors, vol. 18, 2018.

[54] Xu, J., Palanisamy, B., Ludwig, H., Wang, Q., 'Zenith: Utility-Aware Resource Allocation for Edge Computing', IEEE Edge, 2017.

[55] Yang, J., Zhang, H., Ling, Y., Pan, C., Sun, W., 'Task Allocation for Wireless Sensor Network Using Modified Binary Particle Swarm Optimization', IEEE Sensors Journal, 14(3), 2014, pp. 882–892.

[56] Yang, J., 'Complexity Analysis of New Task Allocation Problem Using Network Flow Method on Multicore Clusters', Mathematical Problems in Engineering, art. id 723497, 2014.

[57] Yu, Y., Prasanna, V., 'Energy-Balanced Task Allocation for Collaborative Processing in Wireless Sensor Networks', Mobile Networks and Applications, 10(1-2), 2005, pp. 115–131.

[58] Zhang, Q., Zhani, M. F., Boutaba, R., Hellerstein, J. L., 'Dynamic Heterogeneity-Aware Resource Provisioning in the Cloud', IEEE Transactions on Industrial Informatics, 2(1), 2014.